21世纪高等学校规划教材 | 计算机科学与技术

计算机组织与结构

徐　苏　主编

白小明　张　乐　于海雯　副主编

清华大学出版社

21 世纪高等学校规划教材·计算机科学与技术

计算机组织与结构

徐　苏　主编

白小明　张　乐　于海雯　副主编

清华大学出版社

北　京

内 容 简 介

本书根据我国教育部教学指导委员会制定的“高等学校计算机科学与技术专业规范”组织编写，与美国 ACM 和 IEEE CS 计算机课程最新进展同步，内容涵盖了知识领域 CS-AR 计算机体系结构与组织的核心知识单元和知识点。全书共分 8 章，第 1～7 章全面讲述单处理机系统的硬件组织和结构，包括计算机中的数据表示和运算、汇编级机器组织、存储系统的组织与结构、输入输出系统的组织、CPU 的组织与结构及总线和接口等；第 8 章介绍当前并行处理机系统的一些主流技术和体系结构，包括流水线技术、多处理机系统、机群系统和多核处理器等。

本书是作者根据近二十年计算机组成与计算机体系结构课程教学之经验，并在教学和科研过程中不断积累和提炼而写成的。本书条理清晰，概念准确，所组织的内容不仅全面，而且整合了大量的新技术、新知识，为读者展现近些年来计算机技术发展的新成果。

本书适合作为各类高等院校计算机科学与技术专业的教材，也可作为相关专业工程技术人员和计算机爱好者的参考书。

图书在版编目（CIP）数据

计算机组织与结构/徐苏主编. --北京：清华大学出版社，2015
21 世纪高等学校规划教材·计算机科学与技术
ISBN 978-7-302-39559-1

Ⅰ. ①计…　Ⅱ. ①徐…　Ⅲ. ①计算机体系结构—高等学校—教材　Ⅳ. ①TP303

中国版本图书馆 CIP 数据核字(2015)第 046554 号

责任编辑：刘向威　薛　阳
封面设计：傅瑞学
责任校对：焦丽丽
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　**邮　　编**：100084
社 总 机：010-62770175　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　**印　　张**：22　**字　　数**：533 千字
版　　次：2015 年 5 月第 1 版　**印　　次**：2015 年 5 月第1 次印刷
印　　数：1～　000
定　　价：　.00 元

产品编号：044373-01

出版说明

随着我国改革开放的进一步深化，高等教育也得到了快速发展，各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度，通过教育改革合理调整和配置了教育资源，优化了传统学科专业，积极为地方经济建设输送人才，为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是，高等教育质量还需要进一步提高以适应经济社会发展的需要，不少高校的专业设置和结构不尽合理，教师队伍整体素质亟待提高，人才培养模式、教学内容和方法需要进一步转变，学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007 年 1 月，教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》，计划实施“高等学校本科教学质量与教学改革工程”（简称“质量工程”），通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容，进一步深化高等学校教学改革，提高人才培养的能力和水平，更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部“质量工程”的过程中，各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势，对其特色专业及特色课程（群）加以规划、整理和总结，更新教学内容、改革课程体系，建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上，经教育部相关教学指导委员会专家的指导和建议，清华大学出版社在多个领域精选各高校的特色课程，分别规划出版系列教材，以配合“质量工程”的实施，满足各高校教学质量和教学改革的需要。

为了深入贯彻落实教育部《关于加强高等学校本科教学工作，提高教学质量的若干意见》精神，紧密配合教育部已经启动的“高等学校教学质量与教学改革工程精品课程建设工作”，在有关专家、教授的倡议和有关部门的大力支持下，我们组织并成立了“清华大学出版社教材编审委员会”（以下简称“编委会”），旨在配合教育部制定精品课程教材的出版规划，讨论并实施精品课程教材的编写与出版工作。“编委会”成员皆来自全国各类高等学校教学与科研第一线的骨干教师，其中许多教师为各校相关院、系主管教学的院长或系主任。

按照教育部的要求，“编委会”一致认为，精品课程的建设工作从开始就要坚持高标准、严要求，处于一个比较高的起点上。精品课程教材应该能够反映各高校教学改革与课程建设的需要，要有特色风格、有创新性（新体系、新内容、新手段、新思路，教材的内容体系有较高的科学创新、技术创新和理念创新的含量）、先进性（对原有的学科体系有实质性的改革和发展，顺应并符合 21 世纪教学发展的规律，代表并引领课程发展的趋势和方向）、示范性（教材所体现的课程体系具有较广泛的辐射性和示范性）和一定的前瞻性。教材由个人申报或各校推荐（通过所在高校的“编委会”成员推荐），经“编委会”认真评审，最后由清华大学出版

社审定出版。

目前，针对计算机类和电子信息类相关专业成立了两个“编委会”，即“清华大学出版社计算机教材编审委员会”和“清华大学出版社电子信息教材编审委员会”。推出的特色精品教材包括：

（1）21世纪高等学校规划教材·计算机应用——高等学校各类专业，特别是非计算机专业的计算机应用类教材。

（2）21世纪高等学校规划教材·计算机科学与技术——高等学校计算机相关专业的教材。

（3）21世纪高等学校规划教材·电子信息——高等学校电子信息相关专业的教材。

（4）21世纪高等学校规划教材·软件工程——高等学校软件工程相关专业的教材。

（5）21世纪高等学校规划教材·信息管理与信息系统。

（6）21世纪高等学校规划教材·财经管理与应用。

（7）21世纪高等学校规划教材·电子商务。

（8）21世纪高等学校规划教材·物联网。

清华大学出版社经过三十多年的努力，在教材尤其是计算机和电子信息类专业教材出版方面树立了权威品牌，为我国的高等教育事业做出了重要贡献。清华版教材形成了技术准确、内容严谨的独特风格，这种风格将延续并反映在特色精品教材的建设中。

清华大学出版社教材编审委员会

联系人：魏江江

E-mail：weijj@tup.tsinghua.edu.cn

前　言

1. 计算机学科的课程体系

一个学科的高等教育必须要有先进的教学理念和完整的课程体系，同时与该学科的发展也是紧密相关的。谈到计算机科学与技术学科的高等教育，就要提到两个国际组织，一是IEEE，二是ACM。IEEE的全称是Institute of Electrical and Electronics Engineers，即美国电气与电子工程师学会，是美国的一个工程技术和电子专家的组织，主要致力于电气、电子、计算机工程以及和信息技术与科学有关的领域的开发和研究。ACM的全称是Association for Computing Machinery，即美国计算机协会，是一个致力于工程技术和应用领域中信息技术科学教育的国际计算机组织。国际上最系统、最有影响的计算机专业的教学计划当属IEEE-CS(IEEE属下的计算机学会)与ACM各时期发表的指导性计划，它们对计算机学科教育方面的研究既全面又深入。其中，影响较大的有ACM68课程体系、ACM78课程体系、IEEE-CS83教程和计算机教程1991(简称CC1991)等。ACM68课程体系和ACM78课程体系是基于课程定义的。CC1991是IEEE-CS和ACM合作推出的，它将更多的科学原理引入计算机学科的教学计划设计中，给出了计算机学科的科学定义，解答了计算机学科教育界多年来存在的疑问和争论，同时它采用了知识领域(Knowledge Area)、知识单元(Knowledge Unit)和知识点(Topic)来描述计算机学科的核心知识体系，引导人们去考虑学科的本质和核心，从而制定出既符合各自培养目标又符合学科发展的课程体系。

1998年秋，IEEE-CS和ACM又成立了计算机教程2001(Computing Curricula 2001，CC2001)联合工作组，并于2001年发布了计算机教程2001(即CC2001)。CC2001较好地反映了计算机学科自1991年的发展及这个时期社会发展给学科教育带来的影响。它除了继承CC1991的知识描述体系外，又增加了各级课程的设计方法，并给出了一些推荐课程的描述。

从我国的计算机学科的教育看，1999年前，我国“计算机专业”主要被分为计算机及应用和计算机软件两个专业。从1999年起，按照宽口径培养人才的需要，这两个专业被合并为一个专业，即计算机科学与技术。我国各高校计算机科学与技术专业的教学计划是在中国计算机学会教育专业委员会和全国高等学校计算机教育研究会的指导下制定的。CC2001推出后，中国计算机学会教育专业委员会和全国高等学校计算机教育研究会给予了密切的关注，在对CC2001进行跟踪研究的基础上，结合我国计算机学科的发展现状和我国计算机教育的具体情况，提出了一个适合我国计算机学科教育的课程体系，即中国计算机科学与技术学科教程CCC2002。

近些年，随着计算机技术的高速发展和在各行业应用的普及，社会对计算机学科领域人才的需求分工越来越细，计算机学科的高等教育也发生了变化，各高校在计算机学科先后设置了软件工程、网络工程、电子商务、信息安全等不同的专业或方向，以满足社会对不同专业

人才的需求。从 2001 年开始，IEEE-CS 和 ACM 把计算机学科细分为“计算机科学 CS”、“计算机工程 CE”、“信息系统 IS”、“信息工程 IT”以及“软件工程 SE”。2005 年 IEEE-CS 联合 ACM 发布了 CC2005，2008 年发布了 CS2008 版本（临时版本），2012 年 2 月开始颁布 CS2013。CS2013 重新定义计算机科学的知识模块，考量计算机科学教程的要素，提出一个课程体系样本，发展一系列教学内容，尝试着分别从系统和软件开发两个方面，总结基础原理与方法策略，沉淀计算机学科的专业核心知识。其中对于计算机体系结构与组织（AR）知识模块，强化多核并行、虚拟机支持、供电限制，强调 CAD 工具的应用。

我国教育部高等学校计算机科学与技术教学指导委员会在分析研究计算机学科国内外发展和社会需求的基础上，发布了《计算机科学与技术专业发展战略研究报告》，提出了适应我国计算机科学与技术专业高等教育的教学规范。这一教学规范对于我国各高校在计算机科学与技术专业人才培养目标的确定和课程体系的制定等方面都给出了指导性建议。

2. 为什么要学习本课程

对于学习汽车工程专业的学生来讲，无论是搞汽车外形设计，还是研究汽车的发动机，都必须对汽车的组成和工作原理有基本的了解。同样，对于计算机专业的学生来讲，了解和掌握计算机的组成及工作原理也是必需的。

目前很多高校计算机专业的学生在不同程度上有着重软轻硬的思想。这主要有两个方面的原因：一方面，近十年来，随着各行业管理信息系统建设的发展，社会对软件工程师（尤其是应用软件工程师）的需求越来越大，从事软件设计、软件编程、软件维护等方面的人员成为了 IT 公司、金融、政府及企事业单位紧缺的人才；另一方面，相对软件课程来讲，硬件课程学起来比较枯燥，没有像语言类软件课程有着学完就能使用的立竿见影的效果。例如，很多高校都开设了“Web 程序设计”课程，学生学完该课程后，就能设计网站或制作网页，学生当然很感兴趣。

实际上，在计算机系统中，计算机硬件和计算机软件是相关联的两个部分，硬件为软件的运行提供了一个平台，要编制高质量的软件程序，对计算机有一个整体的了解是十分重要的。对系统软件程序员来讲，系统软件是和硬件紧密相关的，系统软件程序员必须对机器级硬件十分了解，才有可能编制出适应某一机器硬件的系统软件。对应用软件程序员来讲，对机器硬件的了解有助于他们编制更高效和优化的程序。例如，阵列计算机、并行处理计算机、多处理器以及近两年出现的基于多处理器的计算机系统，对并行计算提供了一个支持的平台。对程序员来讲，对计算机硬件实现的并行处理技术的了解，有助于他们充分利用并行计算环境，编制高效的并行程序。

最重要的，前面已经讲到，计算机学科的教育有一个完整的科学体系，课程的设置也是围绕这一体系来进行的。这一科学体系注重培养学生的科学思维能力、创新实践能力、研究和应用能力以及继续学习的能力。作为学生来讲，应该认真学好每一门课程，掌握计算机学科领域所要求的各方面知识。只有这样，才能对本学科有一个完整的理解，才能成为真正合格的计算机科学与技术专业的学生。

计算机组织与结构是计算机专业一门重要的专业基础课程，也是 CC2001、CC2005、CS2013 以及我国计算机科学与技术专业规范中确定的一门核心课程，它对于学生建立计算机整机概念，了解计算机系统的基本组成、结构和工作原理，从而对本学科其他知识领域和

知识单元的内容有更深刻的理解有着非常重要的意义。

3. 教材内容的组织

本教材在内容的组织上，主要按照我国教育部高等学校计算机科学与技术教学指导委员会制定的"计算机科学与技术专业规范"中的知识领域"CS-AR 计算机体系结构与组织"所要求的内容进行编写。同时，紧跟 IEEE-CS/ACM 颁布的 CS2013，对计算机体系结构和组织(AR)知识模块的内容做了适当调整。各章节涵盖的知识单元主要包括

AR2 数据的机器级表示(核心学时)： 第 2 章
AR3 汇编级机器组织(核心学时)： 第 3 章
AR4 存储系统组织与结构(核心学时)： 第 4 章
AR5 接口和通信(核心学时)： 第 5 章、第 6 章
AR6 功能组织(核心学时)： 第 7 章
AR7 多处理和体系结构(核心学时)： 第 8 章

本书共分 8 章，其中，

第 1 章首先介绍计算机的发展历程；然后介绍按 IEEE 分类法的计算机的分类；最后，作为本书的一个"序"，概括性地介绍了计算机的硬件组成及计算机的层次结构。

第 2 章首先介绍二进制等基本的进位计数制；再介绍在计算机中是如何对我们日常处理的数值数据和非数值数据(主要包括字符、汉字等)进行二进制编码表示的；然后介绍数值数据在计算机中的二进制运算方法和实现；最后介绍对计算机中的数据在传递过程中产生的差错进行检测而使用的数据校验码。

第 3 章首先介绍计算机中汇编级指令的格式、地址结构；然后介绍指令及操作数的寻址方式，以及指令的种类和功能、典型指令系统的组成等；最后对精简指令系统 RISC 进行介绍。

第 4 章首先介绍存储器的组织、分类和分层结构；然后介绍计算机主存储器的组成与工作原理；最后介绍提高存储系统性能的交叉存储技术、高速缓冲存储器及虚拟存储器技术等。

第 5 章首先介绍计算机输入输出系统的组成；然后对计算机输入输出的控制方式进行详细讨论，包括程序控制方式、中断控制方式、DMA 控制方式和通道控制方式等；最后介绍计算机存储设备——磁盘系统以及由磁盘阵列组成的 RAID 技术。

第 6 章首先讲述计算机内部各部件之间的总线互连结构，介绍总线的基本概念、总线的类别和总线的控制方式等；然后列举几种现代计算机中常用的 ISA、PCI 等总线标准；最后介绍几种目前在计算机中常用的 USB、IEEE 1394 和 SCSI 等外部总线接口标准。

第 7 章首先介绍 CPU 的功能与组成；再通过一个模型机的例子说明了 CPU 的指令周期及执行指令的过程；然后讨论了 CPU 控制部件设计的硬布线设计法和微程序设计法两种主要方法；最后以 Intel 公司的 CPU 产品为例，介绍了典型 CPU 的发展。

第 8 章首先介绍计算机系统的并行性概念，对计算机中使用的时间重叠、资源重复和资源共享等提高并行性的技术途径进行概要性的介绍；然后分别介绍现代计算机普遍采用的流水线技术和多处理机技术等并行处理技术；最后对近些年发展起来且应用非常广泛的机群系统、高性能计算和多核处理器进行讨论。

4. 教材的主要特色

结合计算机学科教育重基础、重发展、重创新的要求，本教材在内容组织和编写上有以下特点：

(1) 首先为学生建立整机的概念。在第1章，通过将学生日常所熟悉的实际PC与计算机的基本组成部件进行对比，使学生对计算机整机的组成有一个初步的认识，对组成计算机系统的主要部件的基本功能有一个初步的了解。

(2) 按照从整机到部件自上而下的思想进行课程内容的组织，使学生在每一章节的学习中，都清楚所学章节的内容与整机的关联。同时对计算机组织与结构的各种概念、思想和原理等进行重点讲述。

(3) 围绕各章节的内容，穿插了一些"知识拓展"，介绍一些计算机系统方面的相关知识以及计算机发展的新技术、新知识等。如在第1章穿插了知识拓展——摩尔定律，计算机的性能评测；在第2章穿插了知识拓展——二进制的游戏；在第3章穿插了知识拓展——x86CPU指令系统的扩展指令集；在第4章穿插了知识拓展——新型动态存储器SDRAM、DDR、DDR2和DDR3，芯片的封装技术；在第5章穿插了知识拓展——硬盘接口，网络存储系统；在第6章穿插了知识拓展——前端总线，串行传输还是并行传输；在第7章穿插了知识拓展——GPU和CPU；在第8章穿插了知识拓展——超级计算机"天河"二号；能使学生开拓视野，增长知识。

(4) 为帮助学生更好地学习本课程，专门建设了计算机组织与结构课程教学网站。网站的课程介绍部分介绍了本课程的教学大纲、教学的组织和安排等；课程教学部分提供了按课堂教学单元进行组织的教学内容、重点难点等；教学资源部分为学生提供了本课程的教学课件和参考资料；学习讨论部分提供了一个学生与教师以及学生与学生之间的交流平台等。另外，教学管理部分还进一步为教师提供了一个学生管理、作业管理、考试管理和成绩管理等的平台，教师可以通过本课程网站更好地组织本课程的教学。

5. 教材的学时安排

本教材共分8章，第1～7章全面讲述了单处理机系统的硬件组织和结构，包括计算机中的数据表示和运算、汇编级机器组织、存储系统的组织与结构、输入输出系统的组织、CPU的组织与结构及总线和接口等；第8章介绍了现代并行处理机系统的一些主流技术和体系结构，包括流水线技术、多处理机系统、机群系统和多核处理器等。本教材建议总学时为60～80学时，各高校按照计算机专业课程体系中课程设置和讲授内容的不同可以灵活调整。

本教材由徐苏担任主编，白小明、张乐、于海雯为副主编，李向军、林振荣等参编并帮助进行课程教学网站的建设。

编　者

2015年2月

目 录

第1章 计算机系统概述

从 1946 年第一台电子计算机 ENIAC 的诞生到现在，计算机的发展已走过了半个多世纪的历程。在这半个多世纪的时间里，人类在计算机技术领域所取得的成就几乎是其他任何技术领域都无法比拟的。单从衡量计算机性能的重要指标之一——运算速度来看，ENIAC 的运算速度是每秒 5000 次，而现代计算机的运算速度已超过每秒万亿次水平，整整提高了 10^{10} 倍以上。

本章首先介绍计算机的发展历程，其中，器件技术是计算机发展的核心；然后介绍按 IEEE 的分类法的计算机分类和按照计算机的综合性能指标的计算机分类；接着介绍了计算机的主要性能指标；最后，作为本书的一个“序”，概括性地介绍计算机的硬件组成及计算机的层次结构。

1.1 计算机的发展历程

计算机技术的飞速发展离不开其所依赖的硬件技术的发展。在计算机领域，人们普遍把计算机的发展划分为 5 代，而这一划分所依据的正是计算机所使用的基本元器件。可以说，硬件技术是计算机发展的重要物质基础和技术保障。

1. 第零代：机械时代

随着科学的发展，商业、航海和天文学都提出了许多复杂的计算问题，很多人都关心计算工具的发展，希望借助计算工具提高计算的效率，于是人们开始研究和设计具有计算能力的“计算机器(Calculating Machine)”。

世界上第一台以齿轮驱动的计算机器应该是由德国人 Wilhelm Schickard 教授于 1623 年设计并建造的计算钟(Calculating Clock)，如图 1-1 所示。但它并没有得到人们的关注，因为 Wilhelm Schickard 在发明该机器不久就死于疾病。

1642 年，法国数学家、物理学家帕斯卡(Blaise Pascal)在年仅 19 岁时发明了一台机械加法器 Pascaline(见图 1-2)，以帮助其父亲收税时计算使用。由于成本和计算准确度问题，Pascaline 仅售出了 50 台。Pascaline 由一套 8 个可旋转的齿轮系统组成，只能进行加法运算，实现自动进位，并配置一个可显示计算结果的窗口。虽然现在汽车的仪表盘的显示已数字化，但其中里程表中仍然采用了与 Pascaline 相类似的机械工作原理。

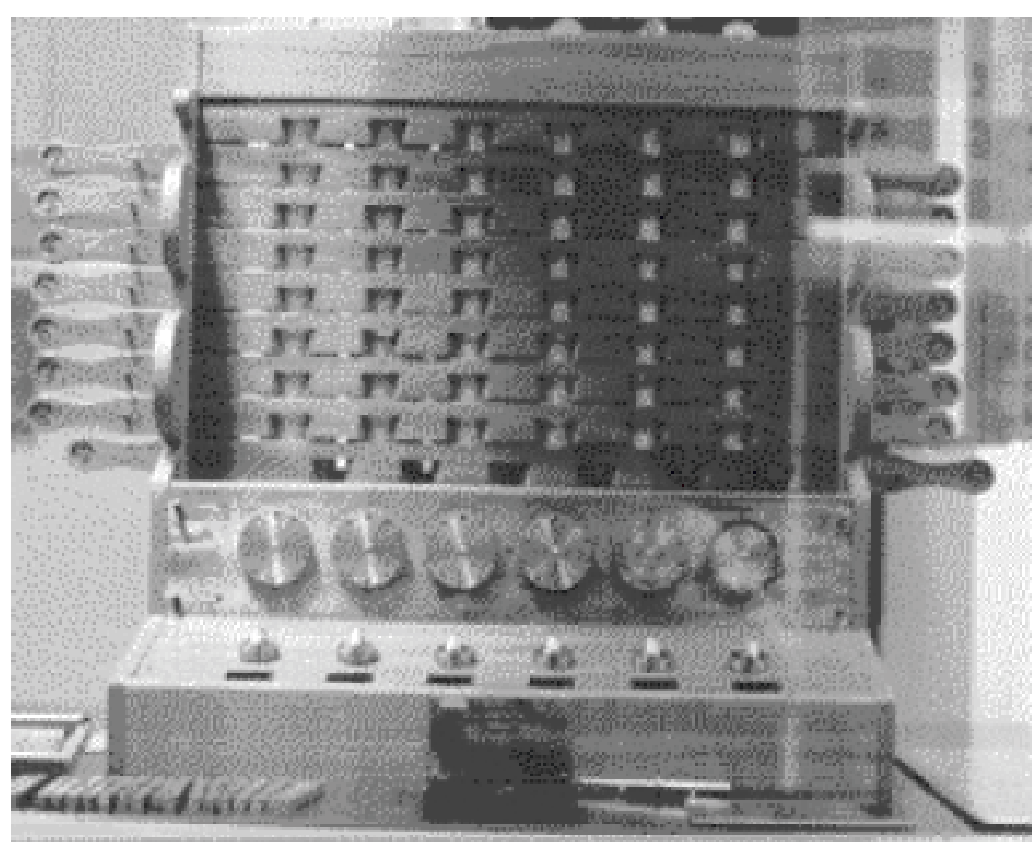


图 1-1　Calculating Clock

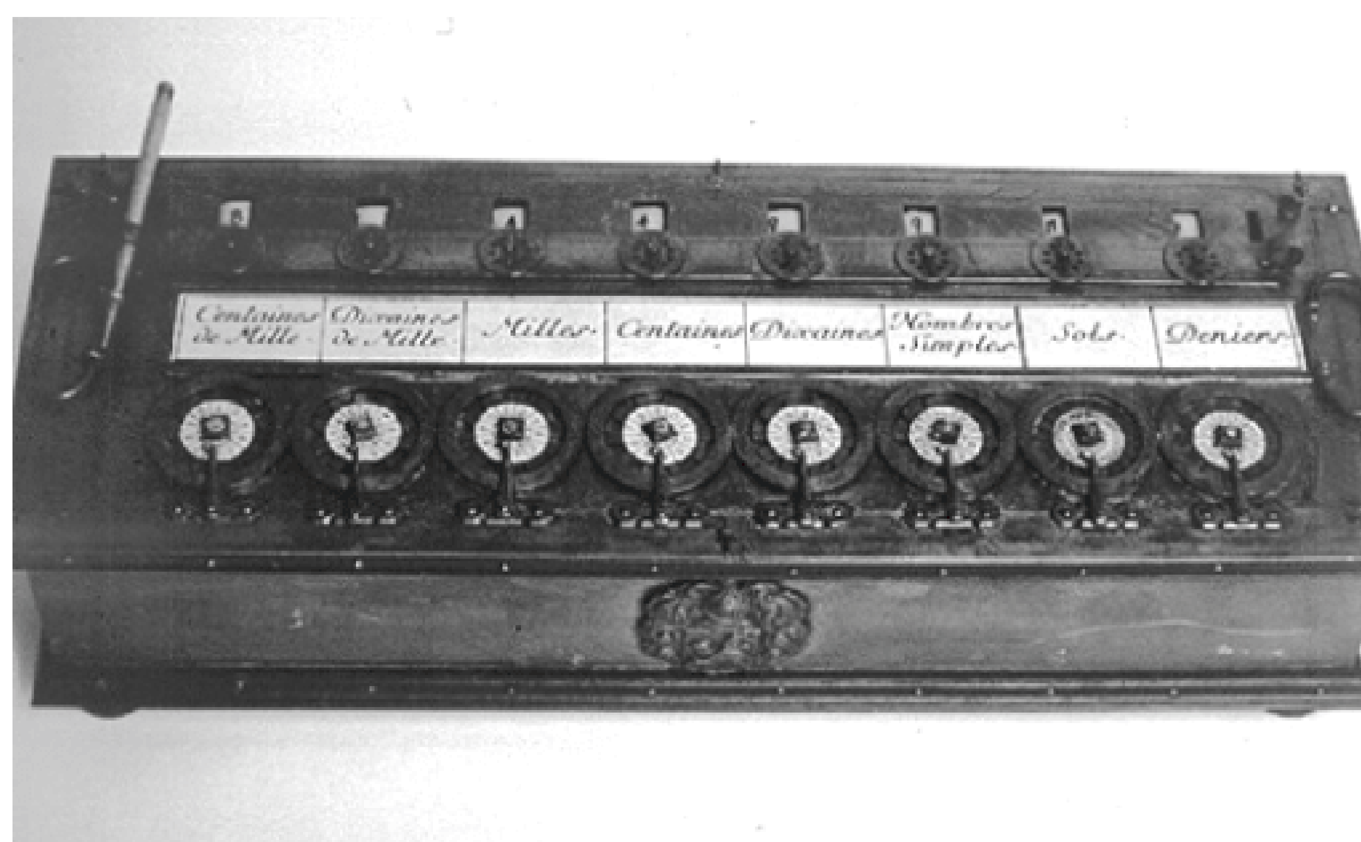


图 1-2　8 齿轮的 Pascaline

1670 年，德国数学家、哲学家莱布尼兹(Gottfried Leibniz)改进了 Pascaline，发明了一个被称为步进式计算器(Stepped Reckoner)的计算器，它具有加、减、乘、除 4 种运算功能。值得一提的是，虽然在 Stepped Reckoner 上使用的是十进制数，但莱布尼兹是首先提出使用二进制计算的科学家，这为现代计算机奠定了基础。但是，所有的这些计算设备或工具都不能进行编程计算，也没有存储器，计算过程中的每一步都需要人手工参与才能进行。

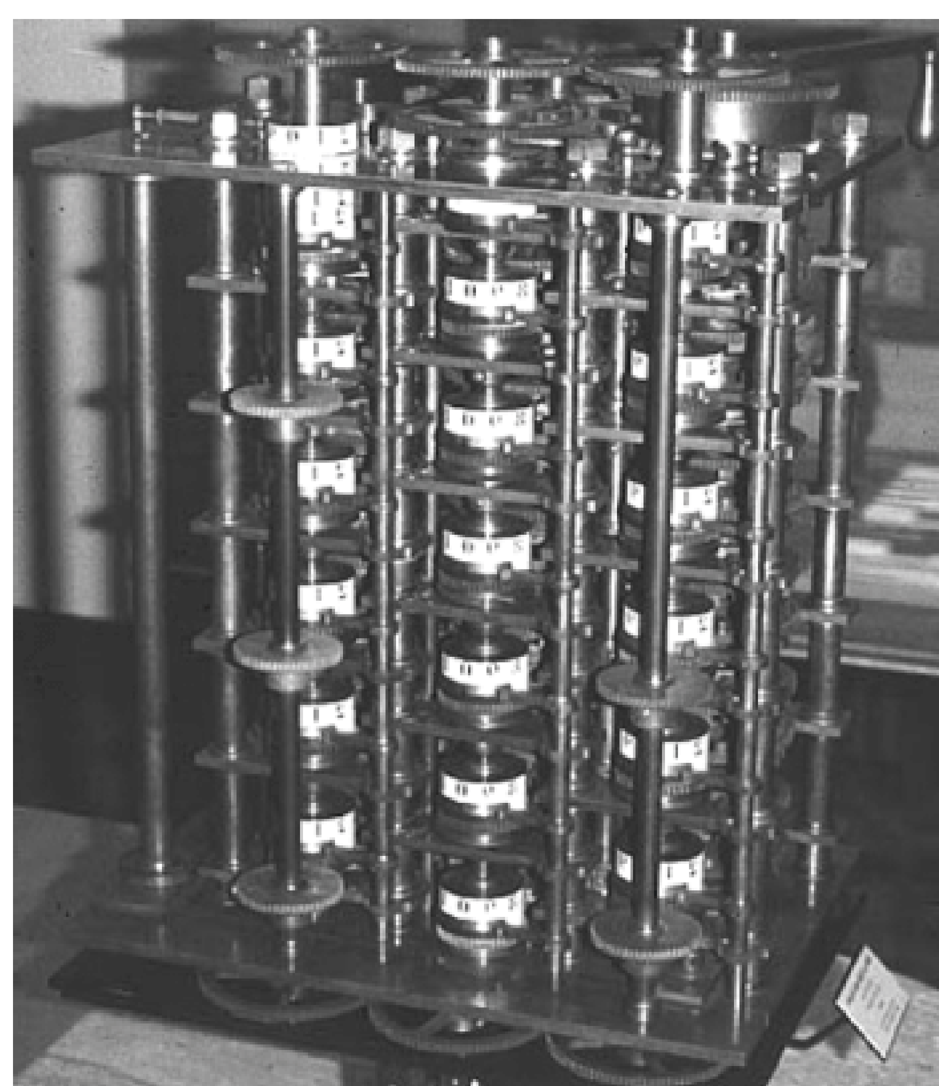


图 1-3　差分机的一小部分机械部件

尽管像 Pascaline 这类计算器一直使用到 20 世纪，但在 19 世纪时就已经开始出现了新型计算工具的设计。这其中，最引人注目的是由英国数学家巴贝奇(Charles Babbage)于 1822 年设计的差分机(Difference Engine)(见图 1-3)。这台机器能够计算数表，如对数表。由于当时数表在航海中的重要性，他得到了英国政府的资助。1833 年，Charles Babbage 又在差分机的基础上设计了一种多用途的机器，称为分析机(Analytical Engine)。分析机已经具备了执行任意类型的数学运算的能力，同时还包含了现代计算机的许多部件：一个算术处理部件进行计算工作(Babbage 称之为运算逻辑部件——mill)，一个存储器(store)，以及输入输出设备。可以说分析机已经具有现代计算机的概念，但因当时的技术条件限制而未能制造完成。

1888 年，美国统计学家霍勒瑞斯(Herman Hollerith)为人口统计局建造了第一台机电式穿孔卡系统——制表机，它是将机械统计原理与信息自动比较和分析方法结合起来的统计分析机，使美国统计人口所需的时间从过去的 8 年缩短为 2 年。霍勒瑞斯在 1896 年创办了制表机公司，1911 年他又组建了一家计算制表记录公司，该公司到 1924 年改名为国际商用机器公司，这就是举世闻名的美国 IBM 公司。

1938 年，德国工程师朱斯(Konrad Zuse)成功制造了第一台二进制计算机 Z-1，它是一种纯机械式的计算装置，它的机械存储器能存储 64 位数。此后他继续研制了 Z 系列计算机，其中 Z-3 型计算机是世界上第一台通用程序控制的机电计算机，它使用了 2600 个继电器，采用二进制进行运算，运算一次加法只用 0.3s，如图 1-4 所示。

1944年，美国麻省理工学院科学家艾肯(Howard Aiken)研制成功了一台通用型机电计算机MARK-Ⅰ，如图1-5所示，它使用了3000多个继电器，总共由15万个元件组成，各种导线总长达到800km以上。1947年，艾肯又研制出运算速度更快的机电计算机MARK-Ⅱ。

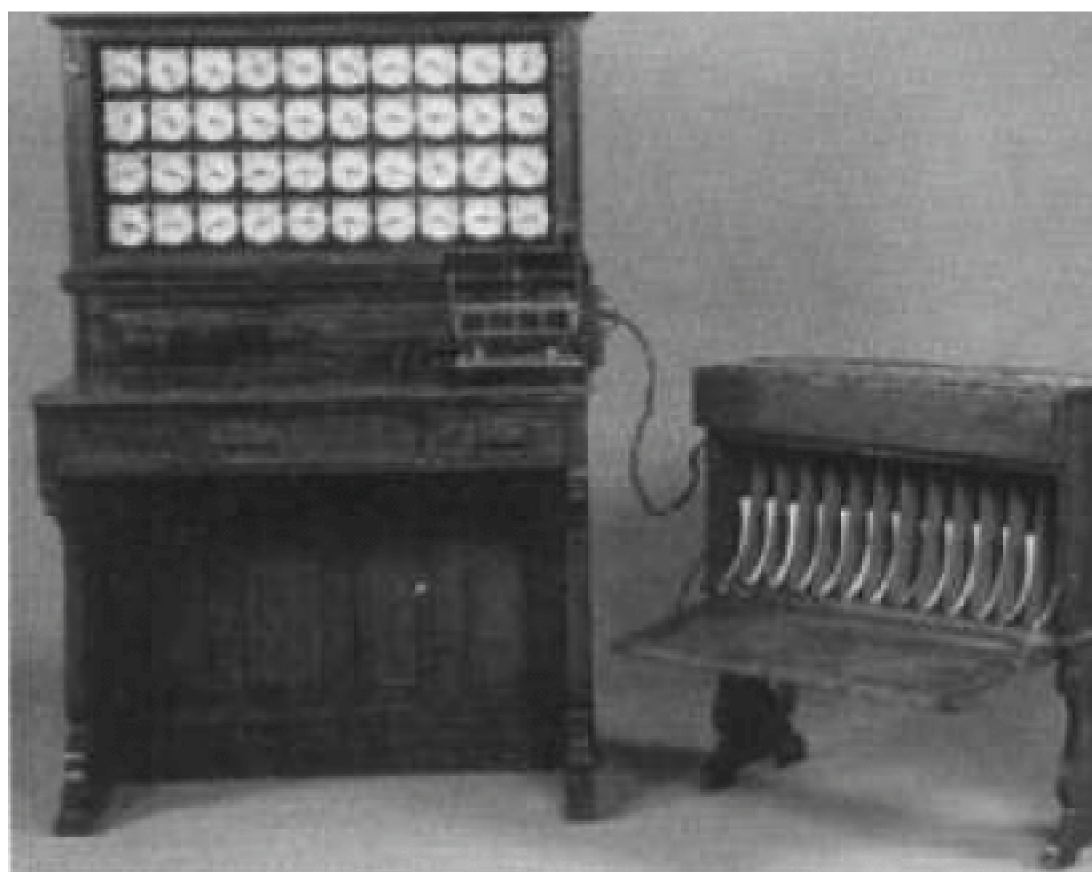


图1-4　Z-3型计算机

至此在计算机技术上存在着两条发展道路，一是各种机械式计算机的发展道路；二是采用继电器作为计算机电路元件的发展道路。后来建立在电子管和晶体管等电子元件基础上的电子计算机正是受益于这两条发展道路的。

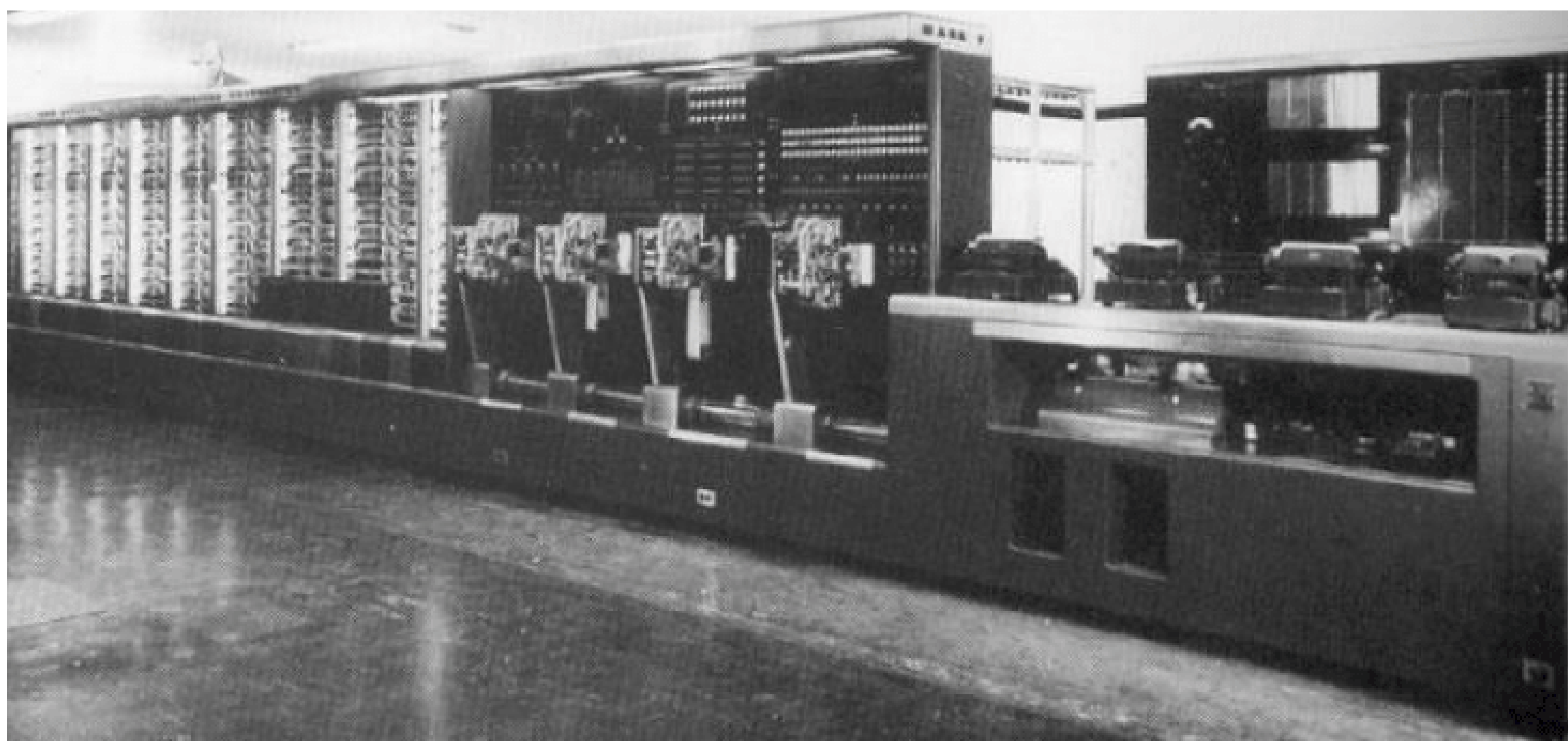


图1-5　MARK-Ⅰ机电计算机

说明：以上图1-1～图1-5的图片取自http://www.computersciencelab.com。

2. 第一代：电子管计算机

电子计算机区别于机械式计算机最主要的特点是使用了电子元器件作为其存储和控制部件，计算机能够依靠电子元器件自动完成计算。电子计算机发展阶段也正是以其所采用的基本电子元器件及技术作为划分的基础，如表1-1所示。

表1-1　计算机的发展阶段

发展阶段	大致时间	所采用的技术	典型速度
1	1946—1957年	电子管	几万次
2	1958—1964年	晶体管	十几到几十万次
3	1965—1971年	中小规模集成电路	百万次
4	1972—1977年	大规模集成电路	千万次
	1978—1991年	超大规模集成电路	亿次
	1991—	巨大规模集成电路	十亿次以上

从表1-1中可以看出，第一代计算机采用的主要元器件是电子管(又称真空管)，它是以1946年诞生的世界上第一台电子计算机ENIAC为标志的，如图1-6所示。

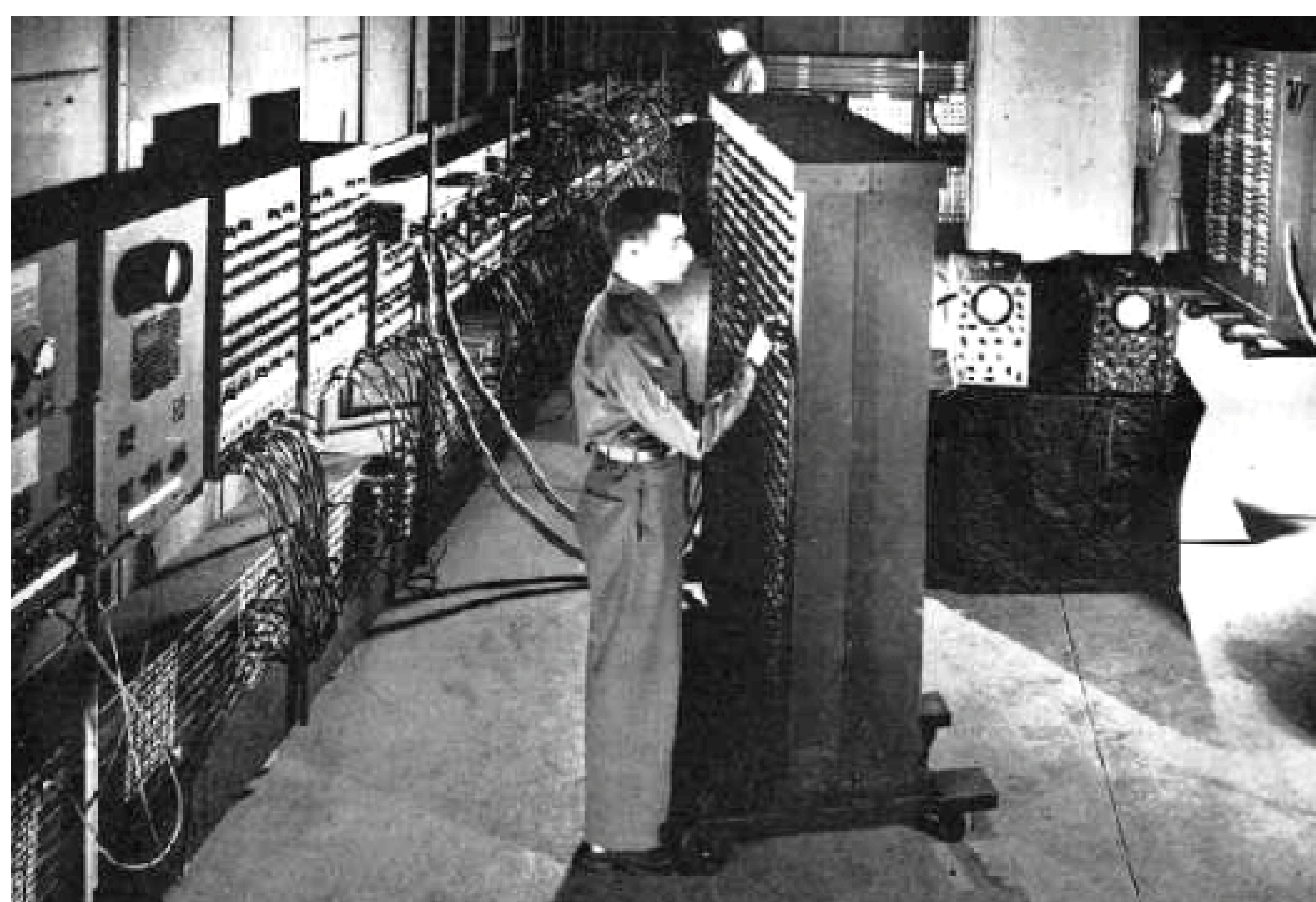


图 1-6　第一台电子计算机 ENIAC

电子数字积分器和计算器(Electronic Integrator And Calculator,ENIAC)是由美国宾夕法尼亚大学电子工程系的 John Mauchly 和 J. Presper Eckert 共同领导设计的。当时正值第二次世界大战期间,美国军方的弹道研究实验室 BRL 在提供导弹发射数据表的精确性和及时性上遇到了困难,军方需雇佣几百人进行人工计算。当得知使用电子计算机可以将导弹发射数据表的计算时间从几天缩短为几分钟时,军方决定资助 ENIAC 项目。ENIAC 项目于 1943 年开始正式启动,1946 年 2 月完成。实际建造出来的机器是一个庞然大物,其占地面积为 $170m^2$,总重量达 30t。机器中约有 18 000 支电子管、1500 个继电器以及其他各种元器件,在机器表面则布满电表、电线和指示灯,每小时耗电量约为 140kW。这样一台“巨大”的计算机,每秒可以进行 5000 次加法运算,相当于手工计算的 20 万倍。计算一条炮弹的轨道用时只需 30s。ENIAC 原来是计划为第二次世界大战服务的,但它投入运行时战争已经结束,这样一来,它便转向为研制氢弹而进行计算。将 ENIAC 用于有别于最初建造它的目的,表明了它的通用性。ENIAC 在 BRL 的管理下继续工作,直到 1955 年被拆除。

第一代电子计算机的主要特点是:

(1) 采用电子管作为基本元器件,体积庞大,功耗大,可靠性低。

(2) 引入存储程序的思想,开始使用存储器存储程序,但最初使用的存储设备为汞延迟线或静电存储管,存储容量很小,后来采用了磁鼓、磁芯,虽有一定改进,但存储空间仍然有限。

(3) 输入输出设备简单,主要采用穿孔纸带或卡片,速度很慢。

(4) 程序设计语言为机器语言,几乎没有系统软件。

在这一时期,人们主要为军事和国防尖端技术的需要研制计算机,并进行有关的研究工作,为计算机的发展奠定了一定的基础,其研究成果进一步扩展到民用,又转为工业产品,开始逐步形成计算机工业。

典型的第一代计算机有 ENIAC、EDVAC、UNIVAC-Ⅰ、IBM 701、IBM 702、IBM 704、IBM 705、IBM 650 等。

3. 第二代:晶体管计算机

事实上,第一代电子管技术的计算机并不是非常可靠,原因是这些电子管被烧坏的速度太快,以至于这种电子管计算机的停机维修时间常常比正常运行的时间还多。

1948 年，贝尔实验室的三位研究员 John Bardeen、Walter Brattain 和 William Shockley 发明了晶体管，这项影响深远的发明，让他们共同获得了 1956 年度诺贝尔物理学奖。这种新型的技术不但掀起了电子器件、电视和无线电广播等领域的革命，也推动计算机的发展进入了一个新的时代。

第二代电子计算机的主要特点是：

（1）采用晶体管代替电子管作为基本元器件。与电子管相比，晶体管具有体积小、重量轻、耗电少、速度快、寿命长等优点，这使计算机的设计结构和性能都产生了飞跃。

（2）采用磁芯存储器作为主存，使用磁盘和磁带作为辅存。使存储容量增大，可靠性提高，为系统软件的发展创造了条件。

（3）提出了操作系统的概念，开始出现汇编语言，并产生了如 COBOL、FORTRAN 等编程语言以及批处理系统。

在这一时期，计算机应用领域进一步扩大，除科学计算外，还用于数据处理和实时控制等领域。同时，这一时期的计算机产品开始重视继承性，形成了适应一定应用范围的计算机"族"，这是系列机思想的萌芽，它可以缩短新机器的研制周期，降低生产成本。

典型的第二代计算机有 IBM 7040、IBM 7070、IBM 7090、IBM 1401、UNIVAC-LARC、CDC 6600 等。

4. 第三代：集成电路计算机

晶体管的采用，使得计算机的体积得以大大缩小，可靠性大大提高。然而随着计算机功能的不断增强，所使用的晶体管数量也随之增加，这在一定程度上又影响了计算机的性能及可靠性，从而限制了计算机的发展。在 20 世纪 40 年代电子管所碰到的应用的窘境，又戏剧性地呈现在晶体管面前。新的现实促使科学家寻找新的解决办法，这就导致第三代电子器件——集成电路的出现。

1952 年，英国皇家雷达研究所的 G. W. A. Dummer 首先提出了集成电路设想：根据电子线路的要求，将电子线路所需要的晶体管、晶体二极管和其他必要元件统统完整地制作（集成）在单块半导体晶片上，从而构成一个具有预定功能的电子线路。但是由于当时缺乏先进的工业手段，Dummer 的设想无法实现。

1958 年，美国的 Jack Kilby 和 Robert S. Noyce 同时发明了集成电路（见图 1-7），并分别在得克萨斯仪器公司和仙童公司研制成功第一块集成电路，从而开创了第三代乃至以后

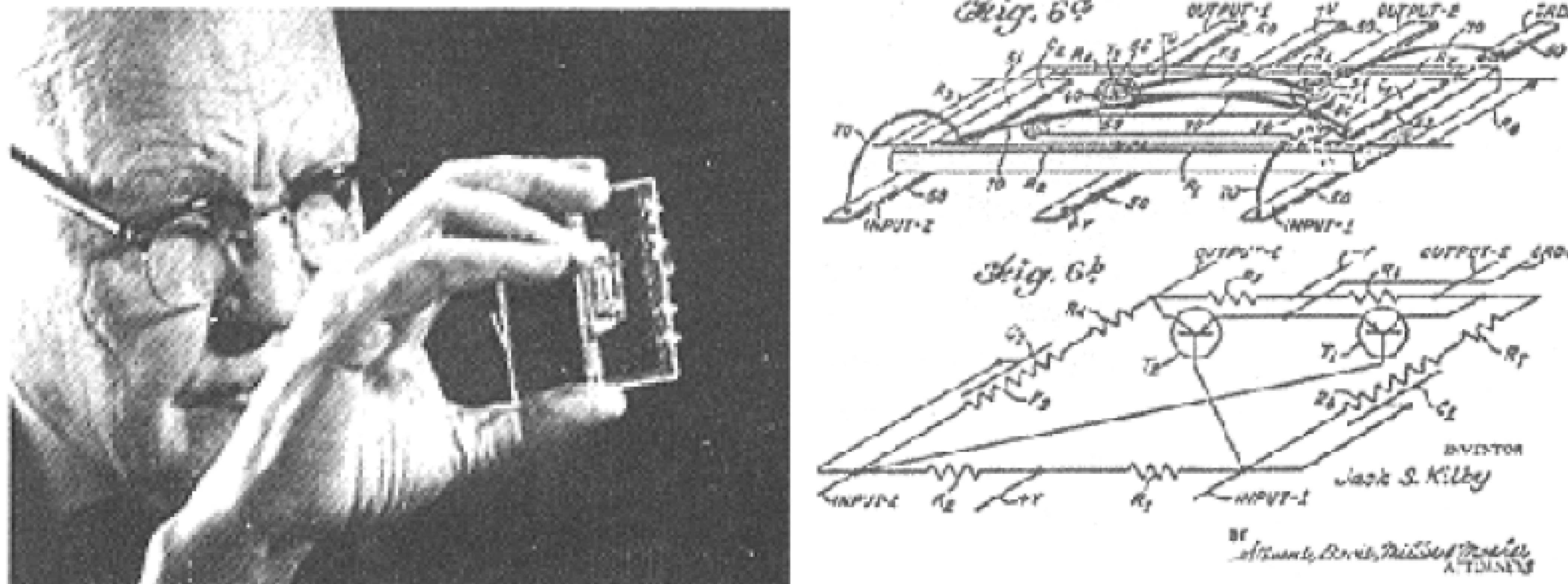


图 1-7　Jack Kilby 和他 1959 申请专利备案的集成电路线路图

计算机发展的新纪元。早期的集成电路虽然只能在单块硅片上集成十几到几十个晶体管，但是，这些集成电路的尺寸已经比分立元件的晶体管还要小。使用集成电路的计算机速度更快、体积更小且价格也更加便宜。

第三代电子计算机的主要特点是：

(1) 采用集成电路取代晶体管作为基本元器件。

(2) 采用半导体存储器作为计算机的主存储器，存储容量进一步扩大，而体积更小，可靠性更高。

(3) 操作系统功能进一步明确和完善，高级语言进一步发展，使计算机功能更强。

(4) 计算机的研制生产开始系列化、通用化和标准化。计算机应用范围扩大到企业管理和辅助设计等领域。

① **系列化**：即同一公司在不同时期生产的计算机采用相同的系统结构，在指令系统、数据格式、字符编码、控制方式、输入输出等方面保持统一，从而保证了程序兼容(称为向前兼容)。这种向前兼容在很大程度上可以保护用户的投资尤其是在软件方面的投资，当用户进行计算机更新时，原来在低档计算机上编写的程序可以不做修改就使用在高档计算机上。这一时期典型的系列计算机如 IBM 360/370。

② **通用化**：机器指令系统丰富，兼顾科学计算、数据处理、实时控制等方面，可以适应各种应用的需要。

③ **标准化**：采用标准的输入输出接口，因而各个机型的外部设备都是通用的，除各个型号计算机的 CPU 独立设计以外，存储器、外部设备等都采用标准部件组装。

典型的第三代计算机有 IBM 360、PDP-Ⅱ、NOVA1200 等。

5. 第四代：大规模和超大规模集成电路计算机

随着芯片技术和半导体制造工艺的不断发展，半导体芯片的集成度越来越高。所谓集成度是指单个芯片上所容纳的晶体管数量。集成电路技术的发展可以分为如表 1-2 所示的几个阶段，其中，大规模集成电路标志着第四代计算机的开始。

表 1-2　集成电路发展阶段

集成电路发展阶段	单块芯片集成的晶体管数量
小规模集成电路(SSI)	10～100
中规模集成电路(MSI)	100～1000
大规模集成电路(LSI)	1000～10 000
超大规模集成电路(VLSI)	10 万以上
巨大规模集成电路(ULSI)	1000 万以上

集成度的提高大大促进了计算机的发展，尤其是进入 20 世纪 70 年代，大规模和超大规模集成电路的发展使计算机进入一个飞速发展的年代。1997 年，为纪念第一台电子计算机诞生 50 周年，一群宾夕法尼亚大学的学生制造了一个单芯片的 ENIAC。原来那个占地 $150m^2$、重 30t 的庞然大物被微缩到只有拇指指甲大小的一块芯片上。

大规模和超大规模集成电路技术的发展使得计算机系统设计者有了很大的施展空间，也使得第四代计算机从各方面性能上讲都有了很大提高，主要体现在以下几个方面：

(1) 在计算机体系结构上，发展了如虚拟存储器技术、流水线技术、高速缓冲存储器技术及各种并行处理技术等。

(2) 在计算机内存配置上，容量和速度都大幅度提高。

(3) 计算机外围设备的种类越来越丰富，从文字的处理发展到图像、声音等多媒体信息的处理。

(4) 系统软件功能越来越强，从单用户、单道程序系统发展到多用户、多道程序系统，从字符界面的命令行操作系统发展到图形界面的 Windows 操作系统。

(5) 应用软件越来越丰富，计算机在办公自动化、数据处理、图像处理、语音识别和人工智能等领域大显身手。尤其是随着微型计算机技术的发展，计算机价格越来越便宜，计算机从机房走向千家万户。

(6) 20 世纪 80 年代计算机网络的发展和 20 世纪 90 年代 Internet 技术的发展，一方面使计算机的应用越来越普及，另一方面在很大程度上改变着人们的生活和工作方式。

值得一提的是 20 世纪 70 年代发展起来的微型计算机技术。1971 年，Intel 公司利用 VLSI 技术制造出世界上第一个微处理器芯片 Intel 4004，第一次将一个 4 位 CPU 的全部电路和功能集成到一块半导体芯片上。第二年，Intel 公司又制造出 8 位的微处理器 Intel 8008，其后又推出了 8080、8085。1978 年和 1979 年，Intel 公司又分别推出了 16 位的微处理器 Intel 8086 和 8088。到了 1981 年，IBM 公司使用 Intel 公司的 8088 微处理器作为 CPU，微软的 MS-DOS 作为操作系统，生产出一台具有里程碑意义的微型计算机 IBM PC，从而开创了微型计算机发展的新纪元。其后，随着微处理器 80286、80386、80486 和 Pentium 等的不断推出，一代又一代新型微型计算机也不断推向市场，直到今天的基于 PIV、酷睿微处理器的计算机。这一切都得益于集成电路技术的发展！表 1-3 给出的是不同时期微处理器芯片的性能参数。

表 1-3 不同时期微处理器芯片的性能参数

型号	Intel 4004	Intel 8008	Intel 8086	Pentium	PIV	Corei7/i5/i3
发布时间	1971 年	1972 年	1978 年	1993 年	2000 年	2008 年
时钟频率	108kHz	108kHz	5MHz	166MHz	1.8GHz	3.3GHz
总线宽度	4 位	8 位	16 位	32 位	64 位	64 位
晶体管数	2300	3500	29 000	3.1 百万	42 百万	700 百万
工艺/μm	10	10	3	0.8	0.18	32nm
可寻址存储器	640B	16KB	1MB	4GB	64GB	64GB
虚拟存储器	—	—	—	64TB	64TB	64TB

6. 第五代计算机

对于前四代计算机，计算机界普遍采用的是按计算机所使用的基本元器件进行划分。即电子管计算机、晶体管计算机、中小规模集成电路计算机和大规模超大规模集成电路计算机。对于人类是否已进入第五代计算机的发展年代，或到底什么样的技术标志着第五代计算机的开始，尚无统一的定论。

在计算机领域，作为第一台电子计算机的诞生国美国，其计算机技术一直处于世界领先水平。日本虽然是工业发达国家，但从计算机整体发展水平看，一直以来落后于美国。然而

日本并不甘落后，1981 年 10 月，日本首先向世界宣告开始研制第五代计算机，并于 1982 年 4 月制订了为期 10 年的“第五代计算机技术开发计划”。紧接其后美国和欧洲等国家也先后提出了发展第五代计算机的计划。

第五代计算机又称新一代计算机或人工智能计算机，它是一个能把信息采集、存储、处理、通信同人工智能结合在一起的智能计算机系统。它除了能进行数值计算及信息处理外，还能面向知识处理，具有形式化推理、联想、学习和解释的能力；能够帮助人们进行判断、决策、开拓未知领域和获得新的知识；人和计算机之间可以直接通过自然语言（声音、文字）或图形图像交换信息。

第五代计算机是为适应未来社会信息化的要求而提出的，与前四代计算机有着本质的区别，是计算机发展史上的一次重要变革。当前电子计算机存在的主要不足有：首先，目前的电子计算机虽然已具有一些相当幼稚的“智能”，但它不能进行联想（即根据某一信息，从记忆中取出其他有关信息的功能）、推论（针对所给的信息，利用已记忆的信息对未知问题进行推理得出结论的功能）、学习（将对应新问题的内容，以能够高度灵活地加以运用的方式进行记忆的功能）等人类头脑最普通的思维活动；其次，目前电子计算机虽然已能在一定程度上配合、辅助人类的脑力劳动，但是，它还不能真正听懂人的语音，读懂人的文章，还需要由专家用电子计算机懂得的特殊的“程序语言”同它进行“对话”。这就大大限制了电子计算机的应用和普及；再次，目前的电子计算机虽然能以惊人的信息处理来完成人类无法完成的工作（如遥控已发射的火箭），但是它仍不能满足某些科技领域的高速、大量的计算任务的要求。例如，在进行超高层建筑的耐震设计时，为解析一种立柱模型受到摇动时的三维振动情况，用目前的超大型电子计算机算上 100 年也难以完成。又如，原子反应堆事故和核聚变反应的模拟实验、资源探测卫星发回的图像数据的实时解析、飞行器的风洞实验、天气预报、地震预测等要求极高的计算速度和精度，都远远超出目前电子计算机的能力极限。由此可见，当今的电子计算机已不能适应信息社会的需要，必须在崭新的理论和技术基础上创造新一代计算机。

当然，让计算机具有像我们人类一样能够进行学习、思维、推理等能力的大脑是一种理想，要实现这一理想还有很长的路要走。但不管怎样，计算机技术日新月异、层出不穷是不争的事实。以下列举一些近些年在计算机领域出现的新的技术：

（1）生物计算机。生物计算机使用生物芯片，生物芯片是用生物工程技术产生的蛋白质分子制成。生物芯片存储能力巨大，运算速度比当前的巨型计算机还要快 10 万倍，能量消耗则为其 10 亿分之一。由于蛋白质分子具有自组织、自调节、自修复和再生能力，使得生物计算机具有生物体的一些特点，如自动修复芯片发生的故障，还能模仿人脑的思考机制。

（2）光子计算机。光子计算机利用光子取代电子进行数据运算、传输和存储。在光子计算机中，不同波长的光表示不同的数据，可快速完成复杂的计算工作。

与电子计算机相比，光子计算机具有以下优点：超高速的运算速度、强大的并行处理能力、大存储量、非常强的抗干扰能力等。据推测，未来光子计算机的运算速度可能比今天的超级计算机快 1000 倍以上。

（3）超导计算机。由超导元件和电路组成的计算机，可依据超导元件的特殊性能而突破电子计算机的局限，使速度更快，消耗更小。

这里再介绍一下我国计算机技术的发展概况。我国从1956年开始研制计算机，1958年研制成功第一台电子管计算机103机，1959年又研制成功运行速度为每秒1万次的104机。103机和104机的研制成功填补了我国在计算机技术领域的空白，为促进我国计算机技术的发展做出了贡献。1965年，中科院计算所研制成功第一台大型晶体管计算机109乙机，之后推出109丙机，该机在我国两弹试验中发挥了重要作用。

1971年我国开始研制以集成电路为主要器件的DJS系列计算机，1974年，清华大学等单位联合设计、研制成功采用集成电路技术的DJS-130小型计算机，运算速度达每秒100万次。该系列计算机在20世纪70年代在我国很多行业和部门得到了广泛应用。

在微型计算机方面，1985年，电子工业部计算所研制成功与IBM PC兼容的长城0520CH微型计算机。此后我国的长城系列、方正系列、联想系列等微型计算机，如雨后春笋般涌现，为我国计算机的普及应用做出了贡献，也使目前以联想公司为代表的中国计算机公司成为了一个有着巨大影响力的跨国公司，走在了世界先进计算机技术的行列。

在巨型计算机研制方面，1983年，国防科技大学研制成功运算速度每秒亿次的银河-Ⅰ巨型机，这是我国高性能计算机研制领域的一个重要里程碑。1992年，国防科技大学又推出了银河-Ⅱ通用并行巨型机，峰值速度达每秒10亿次。银河-Ⅱ为共享主存储器的四处理器向量机，其向量中央处理器是采用中小规模集成电路自行设计的，总体上达到了20世纪80年代中后期国际先进水平，它主要用于中期天气预报、石油勘探和核工业领域等。1997年6月，国防科技大学研制成功银河-Ⅲ百亿次并行巨型计算机系统，采用可扩展分布共享存储并行处理体系结构，由130多个处理节点组成，峰值性能为每秒130亿次浮点运算，系统综合技术达到了20世纪90年代中期国际先进水平。

1995年，曙光公司推出了国内第一台具有大规模并行处理器(MPP)结构的并行机曙光1000(含36个处理机)，峰值速度每秒25亿次浮点运算，实际运算速度上了每秒10亿次浮点运算这一高性能台阶。曙光1000与美国Intel公司1990年推出的大规模并行机体系结构与实现技术相近，这是我国独立研制的第一套大规模并行机系统，打破了外国在大规模并行机技术方面的封锁和垄断，使我国在这一领域的水平与国外的差距缩小到5年左右。1997至1999年，曙光公司先后在市场上推出具有机群结构(Cluster)的曙光1000A、曙光2000-Ⅰ和曙光2000-Ⅱ等超级服务器，峰值计算速度已突破每秒1000亿次浮点运算，机器规模已超过160个处理器。1999年，国家并行计算机工程技术研究中心研制的神威Ⅰ计算机通过了国家级验收，并在国家气象中心投入运行，系统有384个运算处理单元，峰值运算速度达每秒3840亿次。2000年，曙光公司推出每秒3000亿次浮点运算的曙光3000超级服务器。2003年，百万亿次数据处理超级服务器曙光4000L通过国家验收，再一次刷新国产超级服务器的历史纪录，使得国产高性能机产业再上一个新台阶。

2004年，中国研制的超级计算机曙光4000A第一次正式进入国际超级计算机排行榜第10名。2008年，曙光5000A的浮点运算峰值处理能力达到每秒230万亿次，实测Linpack速度达到每秒180.6万亿次，再次跻身世界超级计算机的前10名。2009年10月国防科技大学研制成功天河一号超级计算机，由103台机柜组成，包含6144颗英特尔CPU和5120颗GPU，系统峰值性能为每秒1206万亿次，Linpack测试性能每秒560万亿次，在世界TOP500中排名第五。2010年5月曙光公司和中科院计算所合作研制成功曙光星云超级计算机，系统峰值性能约为每秒3000万亿次，Linpack测试性能每秒1271万亿次，在世界

TOP500 中排名第二。2013 年 5 月，由国防科大研制的天河二号超级计算机系统，以峰值计算速度每秒 5.49 亿亿次、持续计算速度每秒 3.39 亿亿次双精度浮点运算的优异性能位居榜首，成为全球最快超级计算机。值得一提的是，在 2014 年 6 月 23 日公布的全球超级计算机 TOP500 榜单中，中国“天河二号”以比第二名美国“泰坦”快近一倍的速度连续第三次获得冠军。

知识拓展

摩 尔 定 律

摩尔定律是前英特尔联合创始人戈登·摩尔(Gordon Moore)首先提出的，其内容是指：集成电路(IC)芯片中所能集成的晶体管数量每 18 个月翻一番，性能也随之提升一倍。

1965 年，摩尔在为撰写一篇有关集成电路的文章而整理材料和绘制数据时，发现了一个惊人的趋势：集成电路芯片的集成度呈现很有规律的几何增长，每新一代芯片的容量大体为前一代的两倍，而新一代芯片的产生大约为一年的时间。于是摩尔总结得出结论：集成电路芯片中所能容纳的晶体管数量每年增加一倍。摩尔的这一发现发表在当年第 35 期《电子》杂志上，这是他一生中最为重要的文章，这篇不经意之作也是迄今为止半导体发展史上最具有意义的论文。

“当时我在写一篇集成电路的文章，要旨是集成电路技术使电子产品更为便宜。我发现并在文章中描绘了它增长方面的复杂性：一个芯片的容量会逐年递增。从 60 个元件扩展到 64 000 个，每年翻番，而价格上则是相应地逐年递减，当时买一个元件的价格 10 年后可买一个集成芯片，这是一个长期推断。它的事实曲线比我想象得更好”。

摩尔总结的这一规律，由于其可预见性和重要性被正式定义为摩尔定律。1975 年，摩尔对这一结论做了一些修正，将翻番的时间从一年调整为两年。实际上，后来更准确的时间是 18 个月。

“摩尔定律”不是一条简明的自然科学定律，而是一条融自然科学、高技术、经济学、社会学等学科为一体的多学科、开放性的规律。尤其是“摩尔定律”的经济学效益，使其成了英特尔公司的发展指针。

如今人们最关心的就是摩尔定律何时终止。摩尔本人认为它还会延续今后几代产品。“我没有去估算具体的速率，但可能会慢一半左右。翻一番的时间将会是三年而不是 18 个月”。

2006 年 1 月英特尔制造出世界上首款采用 45nm 生产工艺的芯片，该芯片只有指甲大小，但上面却有 10 亿个晶体管，且这些晶体管只有 45nm 见方，比红血球还要小 1000 倍左右。这为今后推出更高性能、更高效率的处理器打下了基础。

2007 年初，IBM 宣布在制造环境中实现了一种突破性的芯片堆叠技术，将传统上并排安装在硅片上的芯片和内存设备以堆叠的方式相互叠加在一起，最终实现了一种紧凑的组件层状结构，大大减小了芯片的体积，并提高了数据在芯片上各个功能区之间的传输速度。这种被称为“穿透硅通道(through-silicon Vias)”的技术可以大大缩小不同芯片组件之间的距离，从而设计出速度更快、体积更小和能耗更低的系统。此举也许会令摩尔定律不仅延续，而且可能突破原来预期的极限。

1.2 计算机的种类

计算机按照其用途分为通用计算机和专用计算机。专用计算机主要是为某一专用领域设计的，在这一领域具有高性能或适应性，如一些专为图像处理设计的计算机，具有很高的图像处理速度；再如工控机，专为一些环境较为恶劣的工业控制领域而设计，具有防尘、防振、高可靠性等特点。通用计算机则不是专为某一领域而设计的，它能适应各种应用的要求。

对计算机种类的划分，更为传统和更为普遍的方法是按照1989年由IEEE科学巨型机委员会提出的运算速度分类法，将计算机分为巨型机、大型机、小型机、工作站和微型计算机。

1. 巨型机

巨型机有极高的速度、极大的容量。主要应用于国防尖端技术、空间技术、大范围长期性天气预报、石油勘探等领域。目前巨型机的运算速度已超过每秒万亿次。这类计算机在技术上朝两个方向发展：一是开发高性能元器件，特别是缩短时钟周期，提高单机性能；二是采用多处理器结构，构成超级并行计算机，通常由100台以上的处理器组成超级并行巨型计算机系统，它们同时解算一个题目，来达到高速运算的目的。

巨型机的研制水平、生产能力及其应用程度，已成为衡量一个国家经济实力与科技水平的重要标志。

2. 大型机

这类计算机具有很强的综合处理能力，性能高，管理能力强。作为通用计算机，主要应用在政府、银行、大企业等部门或行业，作为中心数据库服务器或应用服务器等，可同时支持几十个大型数据库和上万个用户使用。

3. 小型机

小型机相对大型机来说，规模更小、性能也次之。但由于其可靠性高、易于维护等特点，得到了广泛应用。在20世纪七八十年代，小型机的发展非常迅猛，很多行业和部门都使用了小型机，国际上一些大的计算机公司，如IBM、HP、DEC等纷纷推出了自己的小型机系列。从20世纪90年代开始，随着微型计算机的性能不断提高和成本不断降低，小型机市场受到了很大的冲击，一些原来使用小型机的场合被高性能微机服务器所取代。

目前，小型机主要采用的是基于RISC技术的CPU和类UNIX操作系统。

4. 微型机

1971年，美国Intel公司生产出了世界上第一块微处理器芯片Intel 4004，1981年美国IBM公司使用Intel 8088微处理器生产了具有里程碑意义的微型计算机IBM PC，从而揭开了微型机大发展的序幕。微型机技术在短短三十多年的时间里发展迅猛，平均每两年芯片的集成度可提高一倍，从而使得性能提高，价格降低。平均2～3年产品就更新换代一次，几

乎每个月都有相关周边新产品出现。

随着 Internet 的普及，微型机已经像家用电器一样走向了千家万户。

5. 工作站

工作站也是一台独立的计算机，性能一般介于小型机和微型机之间。它往往作为独立的计算机（或联网）应用于一些具有特殊要求的领域，如电影动画特技制作和制造业机械 CAD 就广泛使用了图形工作站。

当然，计算机的分类不是一成不变的。一方面，性能的划分是相对的，例如在某一时期按性能划分为大型机的计算机可能在若干年后还不如一般微型机的性能；另一方面，随着计算机新技术的不断出现，新类型的计算机系统也不断被推出，例如近些年被广泛应用的嵌入式计算机系统就可以看成一种新型的计算机系统。

但是，随着技术的进步，各种型号的计算机性能指标都在不断地改进和提高，以至于过去一台大型机的性能可能还比不上今天一台微型计算机。按照巨、大、小、微、工作站的标准来划分计算机的类型也有其时间的局限性，因此计算机的类别划分很难有一个精确的标准。在此可以根据计算机的综合性能指标，结合计算机应用领域的分布将其分为以下 5 大类。

1. 高性能计算机

高性能计算机也就是俗称的超级计算机，或者以前说的巨型机。目前国际上对高性能计算机的最为权威的评测是世界计算机排名（即 Top 500），通过测评的计算机是目前世界上运算速度和处理能力均堪称一流的计算机。2009 年 10 月国防科技大学研制成功天河一号超级计算机，由 103 台机柜组成，包含 6144 颗英特尔 CPU 和 5120 颗 GPU，系统峰值性能为每秒 1206 万亿次，Linpack 测试性能每秒 560 万亿次，在世界 Top 500 中排名第五。2013 年 5 月，由国防科技大学研制的天河二号超级计算机系统，以峰值计算速度每秒 5.49 亿亿次、持续计算速度每秒 3.39 亿亿次双精度浮点运算的优异性能位居榜首，成为全球最快的超级计算机。

2. 微型计算机

大规模集成电路及超大规模集成电路的发展是微型计算机得以产生的前提。通过集成电路技术将计算机的核心部件运算器和控制器集成在一块大规模或超大规模集成电路芯片上，统称为中央处理器（Central Processing Unit，CPU）。中央处理器是微型计算机的核心部件，是微型计算机的心脏。目前微型计算机已广泛应用于办公、学习、娱乐等社会生活的方方面面，是发展最快、应用最为普及的计算机。人们日常使用的台式计算机、笔记本计算机、掌上型计算机等都是微型计算机。

3. 工作站

工作站是一种高档的微型计算机，通常配有高分辨率的大屏幕显示器及容量很大的内存储器和外部存储器，主要面向专业应用领域，具备强大的数据运算与图形、图像处理能力。工作站主要是为满足工程设计、动画制作、科学研究、软件开发、金融管理、信息服务、模拟仿真等专业领域而设计开发的高性能微型计算机。

需要指出的是，这里所说的工作站不同于计算机网络系统中的工作站概念，计算机网络系统中的工作站仅是网络中任何一台普通微型机或终端，只是网络中的任一用户节点。

4. 服务器

服务器是指在网络环境下为网上多个用户提供共享信息资源和各种服务的一种高性能计算机，在服务器上需要安装网络操作系统、网络协议和各种网络服务软件。服务器主要为网络用户提供文件、数据库、应用及通信方面的服务。

5. 嵌入式计算机

嵌入式计算机是指嵌入对象体系中，实现对象体系智能化控制的专用计算机系统。嵌入式计算机系统是以应用为中心，以计算机技术为基础，并且软硬件可裁剪，适用于应用系统，对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统。它一般由嵌入式微处理器、外围硬件设备、嵌入式操作系统以及用户的应用程序 4 个部分组成，用于实现对其他设备的控制、监视或管理等功能。例如，人们日常生活中使用的电冰箱、全自动洗衣机、空调、电饭煲、数码产品等都采用嵌入式计算机技术。

知识拓展

计算机系统性能评测

对计算机系统性能的评测可以帮助用户进行计算机的选型。国际上曾出现过很多种评测体系，其中，应用面较广泛的包括 TPC 评测体系和 SPEC 评测体系等。

1. TPC 评测体系

事务处理性能委员会（Transaction processing Performance Council，TPC）是由数十家会员公司创建的非营利组织，总部设在美国。TPC 的成员主要是计算机软硬件厂家，而非计算机用户，其功能是制定商务应用基准程序的标准规范、性能和价格量度，并管理测试结果的发布。

TPC 不给出基准程序的代码，而只给出基准程序的标准规范。任何厂家或其他测试者都可以根据规范，最优地构造出自己的测试系统（测试平台和测试程序）。为保证测试结果的完整性，被测试者（通常是厂家）必须提交给 TPC 一套完整的报告，包括被测系统的详细配置、分类价格和包含 5 年维护费用在内的总价格。该报告必须由 TPC 授权的审核员核实。TPC 在全球只有不到 10 名审核员，全部在美国。

2. SPEC 评测体系

SPEC 评测体系由 Standard Performance Evaluation Corp 制定，被引用最广泛的指标主要包括针对 CPU 性能的 SPEC CPU 2000 和针对 Web 服务器的 SPEC Web 2005 等。

SPEC CPU 2000 是一组针对 CPU 和内存的测试，它主要测试的对象是 CPU、内存。SPEC CPU 2000 由许多源代码程序组成，分成“整数”和“浮点数”两组。SPECint 2000 就是“整数”部分，而 SPECfp 2000 则是“浮点数”部分。“整数”部分有 12 个程序，使用 C 或 C++语言，它们不使用 CPU 的浮点单元；而“浮点数”部分有 14 个程序，使用 FORTRAN 77/90 和 C 语言，这些程序的主要运算是浮点数的。

SPECint 2000 和 SPECfp 2000 的结果，以执行时间为准。每个程序的执行时间和一个

参考平台(Sun Ultra5/10 300MHz)相比,计算出其倍数。如果执行时间和参考平台相同,结果就是100。如果只花了一半时间完成,结果就是200。"整数"的12个程序的结果,取其平均值,得到的就是SPECint 2000的测试结果。"浮点数"的14个程序也是一样。

SPEC Web 2005测试的原理是,通过多台客户机向服务器发出Http Get请求,请求调用Web服务器上的网页文件,这些文件从数千字节到数兆字节不等。在相同的时间里,服务器回答的请求越多,就表明服务器对客户端的处理能力越强,系统的Web性能就越好。

1.3 计算机的性能指标

计算机的性能指标是指能在一定程度上衡量计算机优劣的技术指标,计算机的优劣是由多项技术指标综合确定的,而这些指标主要包括吞吐量、响应时间、CPU时钟周期、主频、CPI、CPU执行时间、MIPS、MFLOPS、GFLOPS、TFLOPS、PFLOPS等。

1. 吞吐量

在一个评价期间内,计算机系统完成的所有工作负载称为吞吐量。

2. 响应时间

响应时间指用户输入一个作业(或事务)至输出开始之间的间隔时间。

3. 主频/CPU时钟周期

CPU的主频是CPU内核工作的时钟频率,一般以MHz或GHz为单位。很多人认为CPU的主频就是其运行速度,其实不然。CPU的主频表示在CPU内数字脉冲信号振荡的速度,与CPU实际的运算能力并没有直接关系。主频和实际的运算速度存在一定的关系(在同一个系列计算机中,在同样条件下,主频越高,速度越快),但目前还没有一个确定的公式能够定量两者的数值关系,因为CPU的运算速度还要看CPU的流水线各方面的性能指标(缓存、指令集、CPU的位数等)。

CPU时钟周期是一个时间的量,一般以微秒或纳秒为单位。主频的倒数就是CPU时钟周期,这是CPU中最小的时间元素。每个操作至少需要一个时钟周期。

4. CPI

每条指令执行需要的时钟周期数(Cycle Per Instruction,CPI)指CPU的指令时钟数。表示每条计算机指令执行所需的时钟周期数。由于不同指令的功能不同,造成指令执行时间不同,即指令执行所用的时钟数不同,所以CPI应该是一个平均值。

5. CPU执行时间

CPU执行时间是指CPU全速工作时完成某进程所花费的时间,计算公式如下:

$$\text{CPU 执行时间} = \frac{\text{CPU 时钟周期数}}{\text{时钟频率}} \tag{1.1}$$

6. MIPS 和 MFLOPS、GFLOPS、TFLOPS、PFLOPS

每秒执行多少百万条指令(Million Instructions Per Second,MIPS)定义为

$$\text{MIPS}=\frac{\text{指令条数}}{\text{执行时间}\times 10^{-6}} \tag{1.2}$$

每秒执行多少百万次浮点运算(Million FLoating-point Operations Per Second,MFLOPS)定义为

$$\text{MFLOPS}=\frac{\text{浮点操作次数}}{\text{执行时间}\times 10^{-6}} \tag{1.3}$$

$$\text{GFLOPS}=\frac{\text{浮点操作次数}}{\text{执行时间}\times 10^{-9}} \tag{1.4}$$

$$\text{TFLOPS}=\frac{\text{浮点操作次数}}{\text{执行时间}\times 10^{-12}} \tag{1.5}$$

$$\text{PFLOPS}=\frac{\text{浮点操作次数}}{\text{执行时间}\times 10^{-15}} \tag{1.6}$$

1.4 计算机的基本组成

计算机经历了几十年的发展,虽然性能越来越高,适应各种应用的产品越来越丰富,但从其基本的硬件组成上讲,仍然是采用当初的冯·诺依曼(von Neumann)结构进行设计的。无论是巨型机、大型机还是小型机、微型机,它们主要的不同都在于性能的高低,而从硬件组成上讲是基本相同的。

那么,一台完整的计算机主要由哪些硬件组成呢?以微型机为例,通过对其进行彻底“解剖”,看看日常使用的计算机中包含了哪些部件。

图 1-8 是 1981 年 IBM 公司生产的一台具有里程碑意义的 IBM PC(今天讲的 PC 或个人电脑正是从该计算机的名字得来的)。之所以说它具有里程碑的意义,一方面是因为它的推出彻底改变了人们使用计算机的工作模式,使计算机从神秘的空调房走到了办公桌,走向了家庭;另一方面,因为它的廉价和易于维护,使计算机得以普及,使计算机的应用得到了前所未有的发展。虽然,从今天的角度来看,IBM PC 的配置会让人觉得不可思议,这种配置的计算机能干什么?但在当时,它确实让很多人激动。

- Intel 8088 CPU,4.77MHz
- 64KB内存
- 640×480分辨率单色显示器
- 5.25in软驱
- 键盘
- DOS操作系统

图 1-8 1981 年生产的 IBM PC

再来看看我们今天使用的 PC(如图 1-9 所示)。多核 CPU,DDR 内存,SATA 硬盘,液晶显示器,光电鼠标,人体工程学功能键盘,音响系统,Windows 8 操作系统,等等,各种先进技术让人眼花缭乱。

- Intel酷睿i7/i5/i3 CPU
- 4GB DDR3内存
- 1TG 7200转高速 SATA硬盘
- 19″宽屏液晶显示器
- 10/100/1000M集成网卡
- 光电鼠标/人体工学程功能键盘
- 高品质音响系统
- Windows 8操作系统

图 1-9　现代 PC

从 IBM PC 到现代 PC,虽然只有短短三十几年的时间,计算机技术却取得了突飞猛进的发展,性能已得到了大大提高。然而,从专业角度看,它们的硬件组成及功能结构却是相同的。下面以现代 PC 为例,对其进行进一步的剖析。

从外观看,一台台式 PC 至少包括机箱、显示器、键盘和鼠标等(有的 PC 还会配上音箱系统)。打开机箱,里面还有电源、主板、硬盘、光驱、显卡、网卡等,主板上又有 CPU、内存等,这么多的东西让人眼花缭乱。图 1-10 是实际主板的构成图。

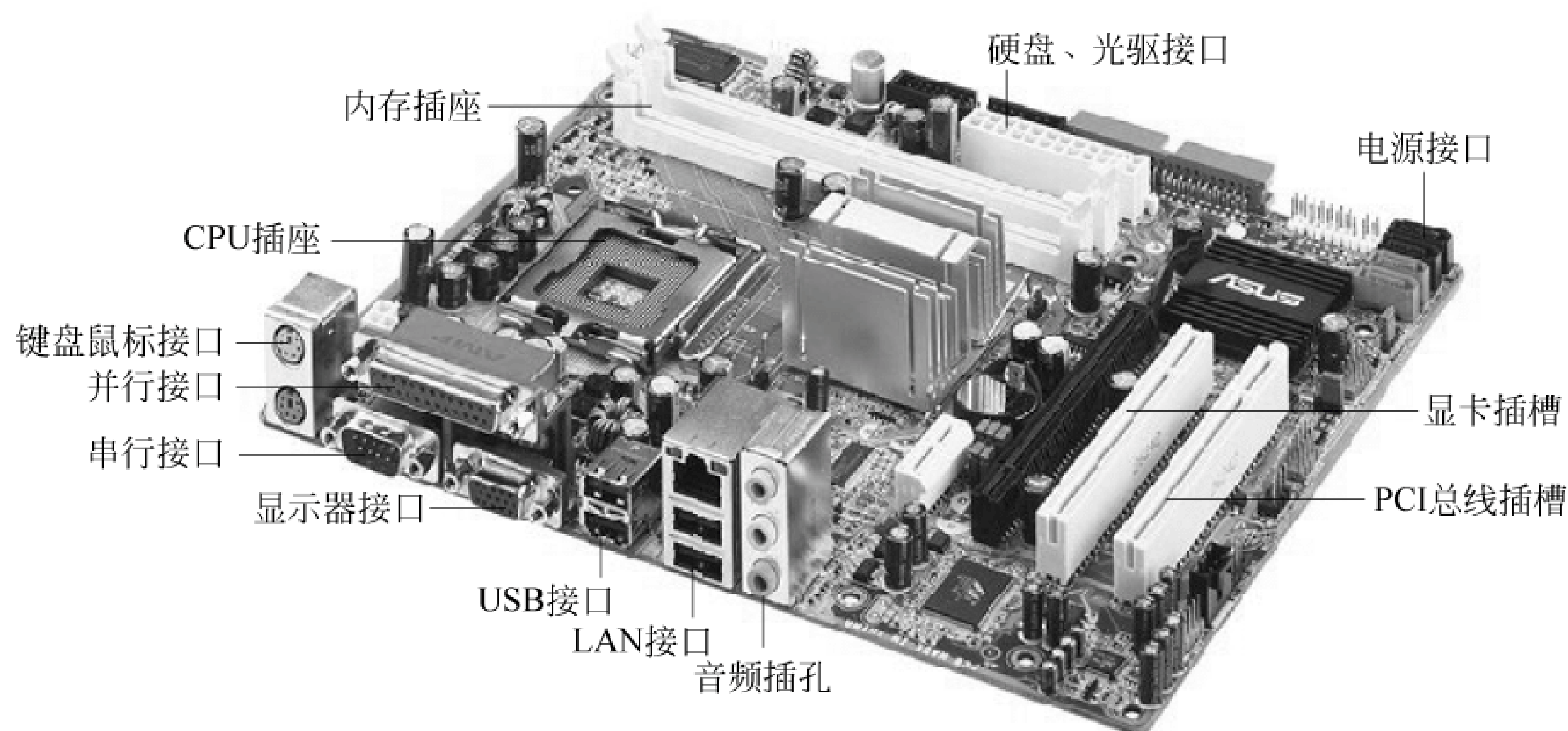


图 1-10　主板构成图

将以上所罗列的设备及部件整理一下,可以得到如图 1-11 所示的计算机硬件组成结构图。

从图 1-11 中可以看出,从功能结构上讲,计算机硬件主要由 CPU、存储器、输入输出接口及设备三大部分组成。而从设计和实现上讲,CPU、存储器(目前在计算机中是以内存的方式出现的)直接插在主板上,同时在主板上还集成了一些输入输出接口电路。显示器作为标准输出设备,键盘和鼠标则作为标准输入设备,分别与主板上对应的接口相连。另外,磁盘(包括软盘和硬盘)、光盘等从功能上讲属于计算机的辅助存储器(第 4 章详细介绍),但从操作系统的角度上讲,是将它们作为输入输出设备进行管理的。

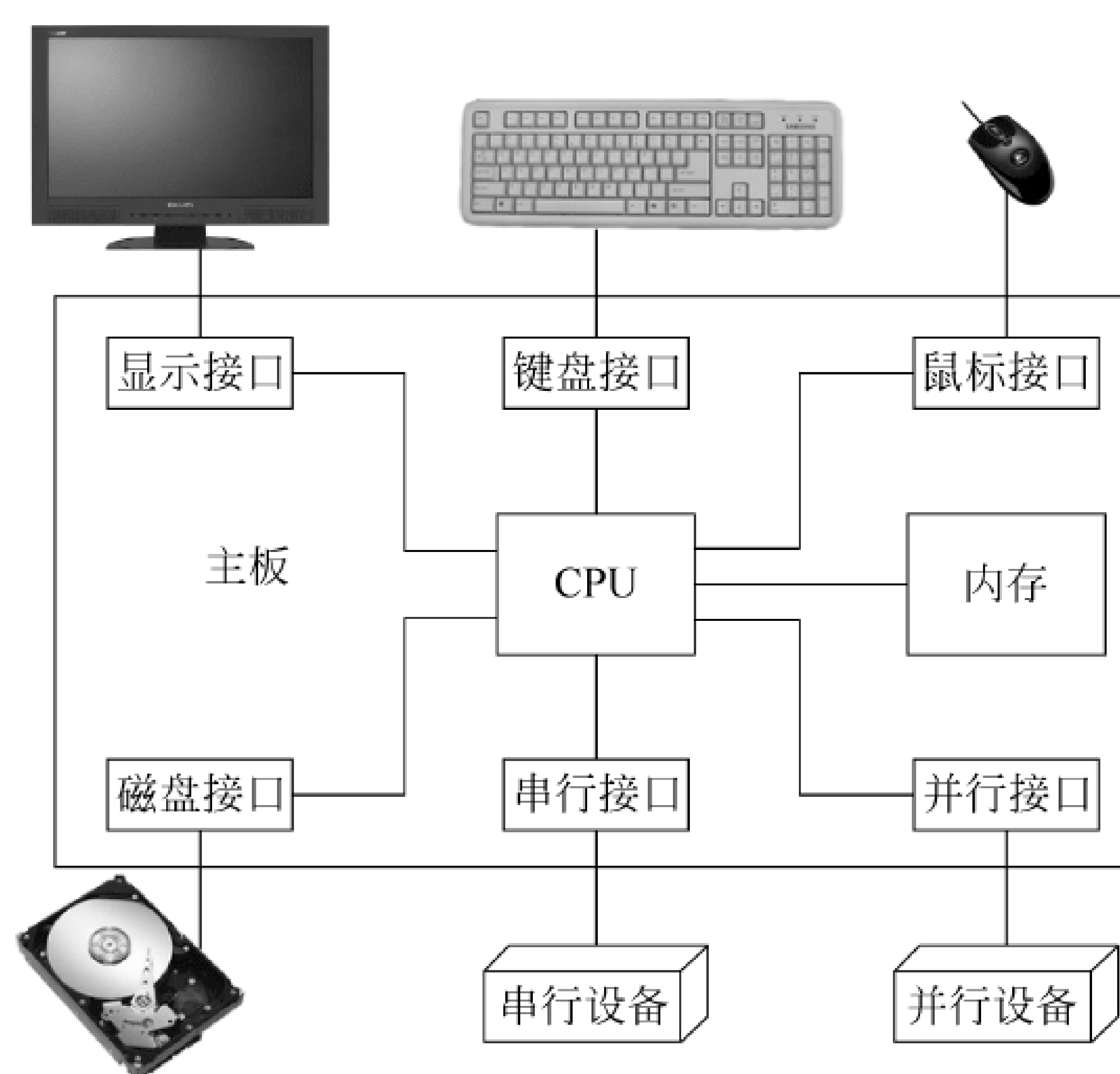
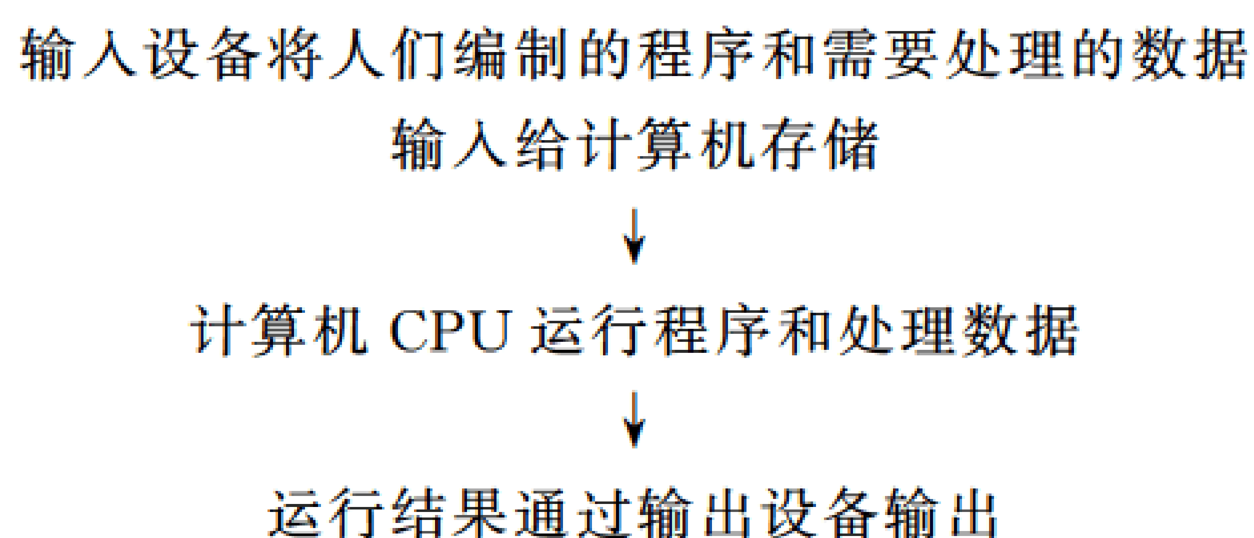


图 1-11 计算机硬件组成结构

其实,计算机的这种组成结构也正符合其“计算”的要求。人们为解算某一问题编制程序,然后交给计算机运行程序,并最终将运行结果返回,这一过程正需要不同的设备或部件配合完成。



另外,CPU、存储器和输入输出接口及设备间需要进行数据的通信,因此,它们之间需要通过某种方式连接起来。在现代计算机中,普遍采用的是一种总线连接方式,如图 1-12 所示。

总线是一组信号线,它主要包括 CPU、存储器和输入输出接口及设备间需要进行传输的数据信号、地址信号和控制信号等。采用总线连接方式,可以有效减少这些部件之间重复连接的信号线的数量和提高通信的效率。有关总线的内容将在第 6 章详细介绍。

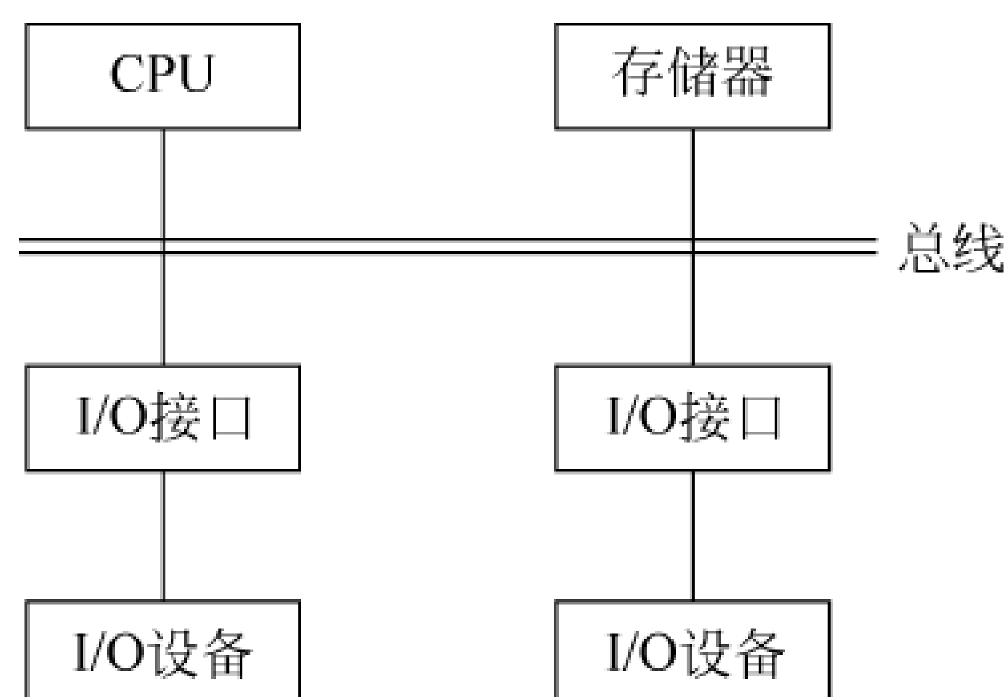


图 1-12 计算机硬件之间的总线连接

在计算机的各个组成部件中,CPU 是最核心的部件,计算机的各个部件所进行的所有操作和运算都是在 CPU 控制下完成的。而从功能结构上讲,CPU 又是由运算器和控制器两大部分构成的。因此又可以说,计算机由五大部件构成,即运算器、控制器、存储器、输入部件和输出部件。

1. 运算器

计算机的一个最大特点就是具有很强的运算能力。运算器就是计算机内用于完成各种运算的部件，其基本结构如图 1-13 所示。

运算器中有一个核心部件——算术逻辑运算单元(ALU)，它能完成各种算术运算和逻辑运算。最基本的算术运算主要包括加、减、乘、除等，逻辑运算主要包括与、或、非、异或及移位运算等。通过这些最基本的算术、逻辑运算可以实现各种复杂的运算。ALU 一般有两个输入端，它能一次完成两个操作数的运算。另外在运算器中还会设置一些通用寄存器 R，用于暂时存放运算中产生的中间结果。

2. 控制器

控制器是计算机的指挥中心，它按照人们预先编好的程序进行工作，根据程序中指令的要求，有序地向计算机中各个部件发出控制信号，使计算机中各个部件有条不紊地工作，从而完成指令所要求的功能。控制器的基本结构如图 1-14 所示。

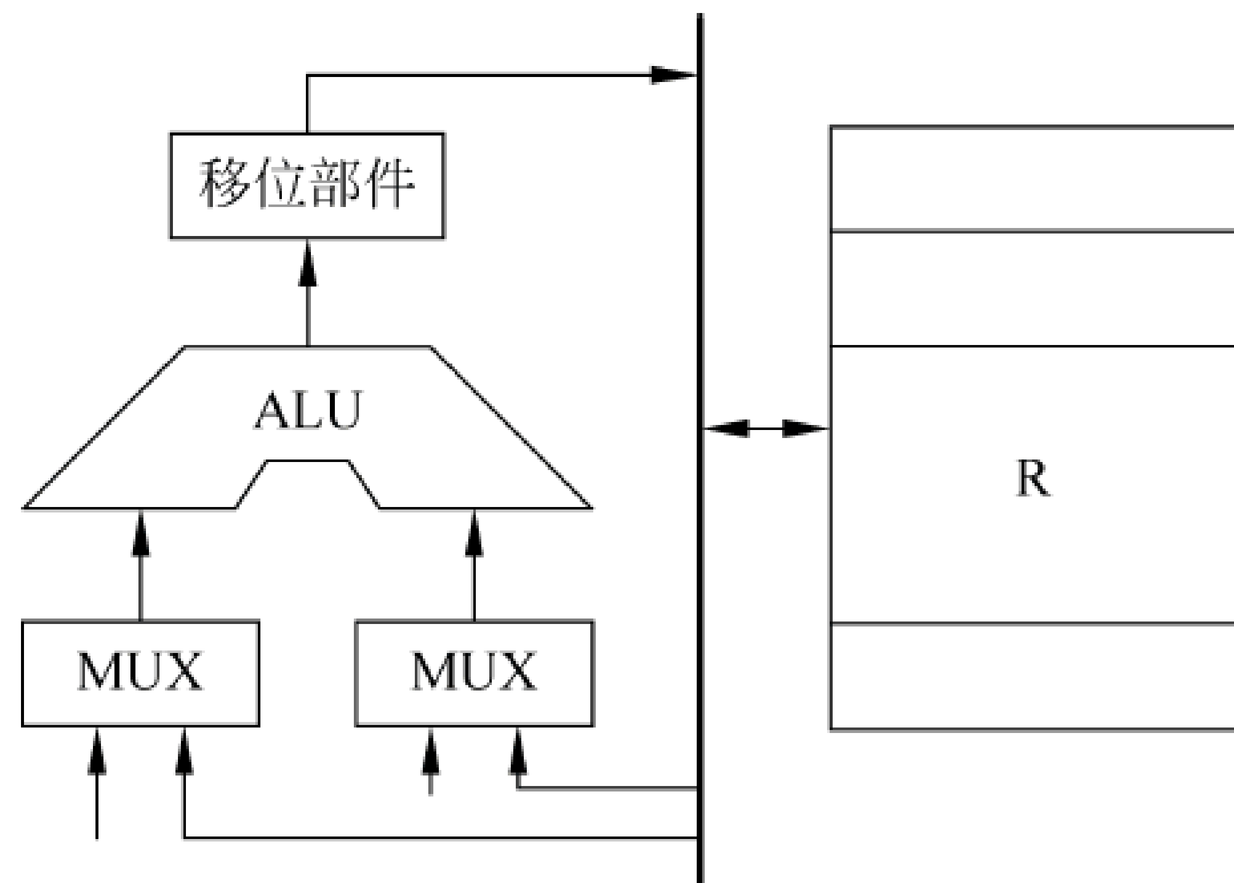


图 1-13 运算器结构

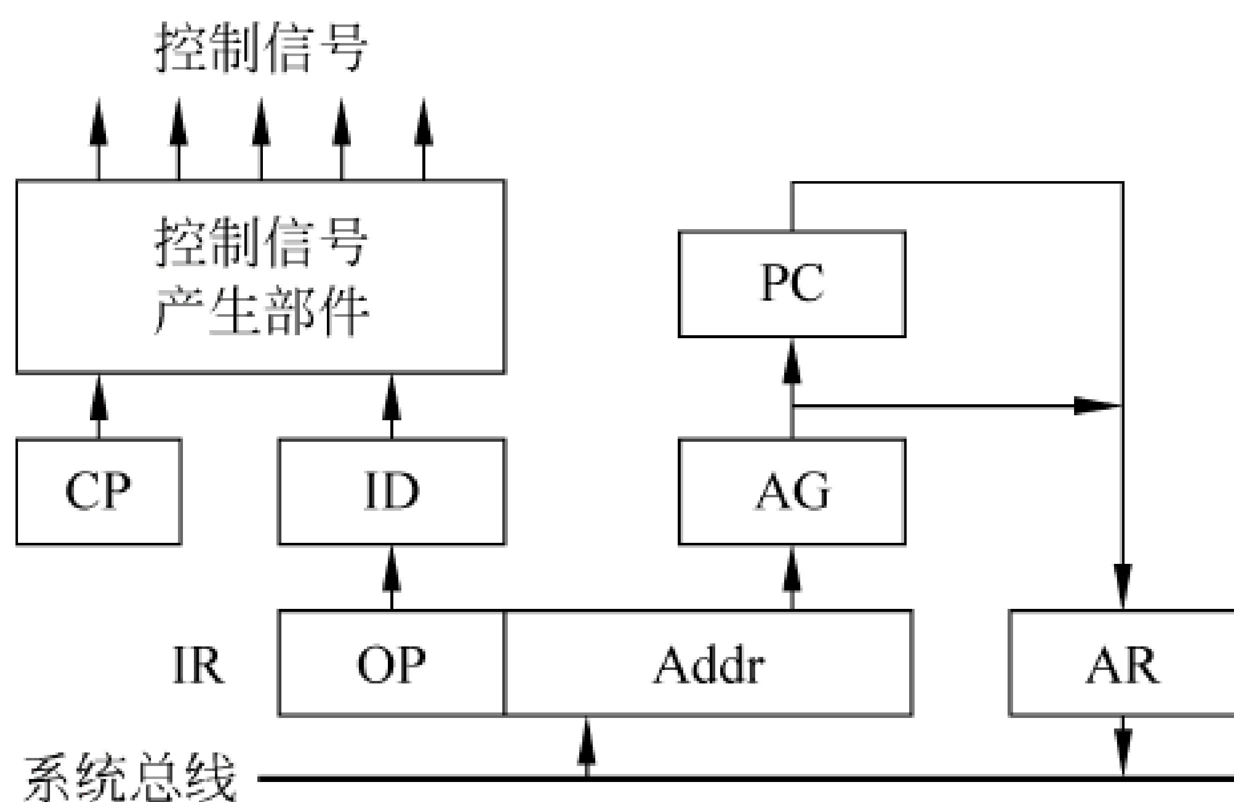


图 1-14 控制器结构

在控制器中主要包括程序计数器(PC)、指令寄存器(IR)、指令译码器(ID)、地址生成器(AG)、地址寄存器(AR)、时序部件(CP)和控制信号产生部件等。

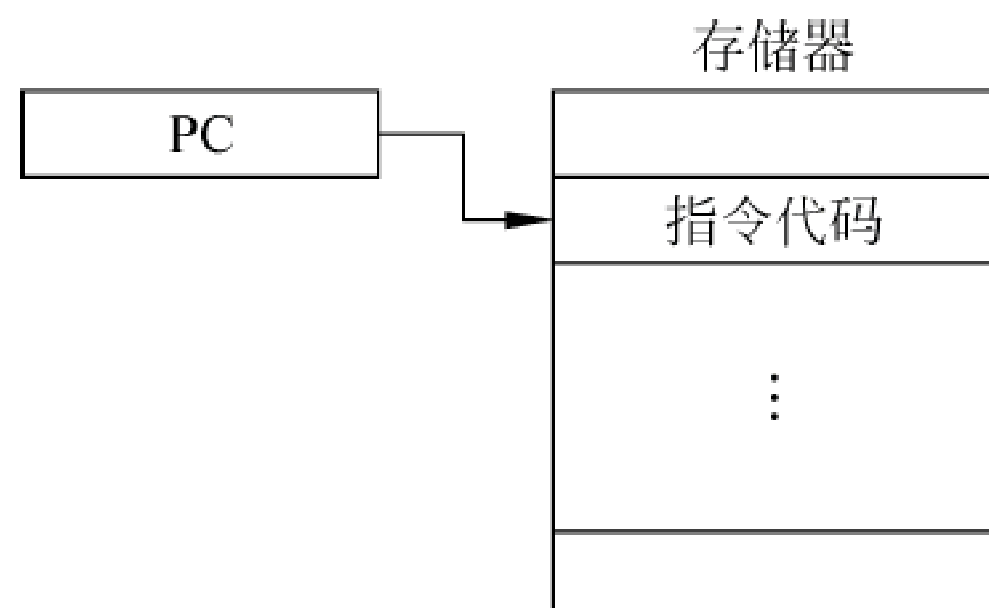


图 1-15 PC 的功能

程序计数器(Program Counter，PC)实际上是一个地址寄存器，其中存放的是下一条要执行的指令在存储器中的单元地址(见图 1-15)。按照冯·诺依曼存储程序的思想，要执行的程序指令是事先存放在存储器中的(这里指的是计算机主存储器)，然后由 CPU 从存储器中一条一条将指令取出并执行。程序的执行是在操作系统的控制下完成的。当要运行一个程序时，首先由操作系统将该程序从计算机外部存储器调入主存储器中，然后将为程序动态分配的存储单元首地址(程序要执行的第一条指令所在存储单元的地址)送入 PC 中，这样就开始了该程序的执行。程序在存储器中一般是按其编写的顺序存放的，若当前执行的指令是一条顺序指令，则由 PC 加 1 自动生成下一条指令的地址，而若当前执行的指令是一条转移或转子指令时，则由该指令生成要转向的指令地址并送 PC。

指令寄存器(Instruction Register,IR)主要用于存放由PC指向的从存储器中取出的指令代码。指令寄存器一般为一个指令字长,它主要由两个字段组成,一是指令操作码OP字段;二是操作数或转移地址Addr字段。OP字段用于指出该指令是一条什么样的指令,如加法、移位等;Addr字段根据指令的不同功能有所不同,对于顺序指令,Addr字段用于指出操作数的类型及存放的位置或地址,而对于转移或转子指令,Addr字段则用于指出要转向的指令的地址。

指令译码器(Instruction Decoder,ID)用于对指令寄存器中的OP字段进行译码,并将译码结果输出给控制信号产生部件,如图1-16所示。例如,假设某计算机共有64条指令,采用定长操作码编码,即所有指令的OP字段均为6位二进制,则该计算机的指令译码器的一种简单设计就是采用一个6/64译码器,即6位输入,产生64个输出,且同时只有一个输出有效。在进行指令系统设计时,为每一条指令分配一个不同的6位二进制代码。当某条指令执行时,译码器的输入是该指令的6位OP代码,则其对应的输出有效,此输出用于“通知”控制信号产生部件当前执行的是该指令。

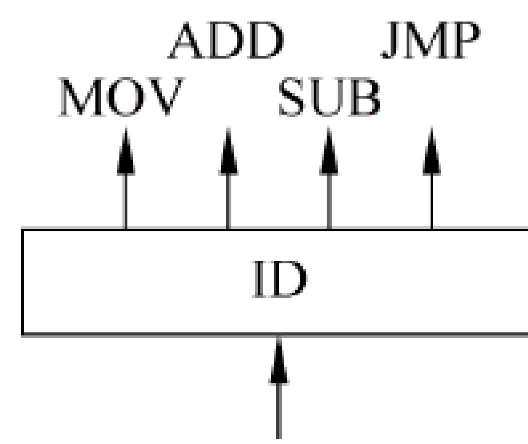


图1-16 译码器功能

地址生成器(Address Generator,AG)主要用于生成操作数在存储器中的单元地址,从而为取该操作数做好准备。为了编程的灵活性,几乎所有计算机的指令系统都提供了多种操作数的寻址方式,不同的寻址方式生成操作数的实际地址的方法有所不同,设置地址生成器的目的就是为了支持各种寻址方式的实现。有关寻址方式的内容将在第3章详细讲述。

地址寄存器(Address Register,AR)主要有两个用途:一是用于存放由地址生成器按寻址方式进行计算得到的操作数在存储器中的地址;二是用于存放由当前转移或转子指令产生的要转向的指令的地址。对CPU来说,无论是从存储器中取指令还是取操作数,都属于访存操作。访存操作的第一步就是要给出所要访问的存储单元地址,而该地址就是由地址寄存器保存的。

最后,所有指令的执行都是在一定的操作控制信号的控制下完成的,操作控制信号产生部件就是根据指令译码的结果产生当前指令执行过程中所需的全部操作控制信号,这些控制信号在时序部件产生的时序下按照一定的顺序逐个产生,从而控制不同的部件完成相应的操作。

3. 存储器

存储器是计算机重要的组成部件之一,主要用来存储程序和数据。现代计算机普遍采用冯·诺依曼计算机结构,而冯·诺依曼计算机的特点之一就是“存储程序”。所谓存储程序是指:任何要运行的程序必须事先存入存储器中,由CPU从存储器中将程序指令一条条取出并执行。因此,存储器是现代计算机不可或缺的部件。

从物理上讲,存储器是由具有一定记忆功能的物理器件构成的,如目前计算机内存普遍采用的半导体存储器和计算机外存采用的磁介质存储器或光存储器等。这些物理器件是通过自身的一些物理特性来表示和存储二进制0、1信息的。例如,在第4章将会讲到的一种静态半导体存储器,它的每个存储单元电路(存储1位二进制信息)是依靠三极管的导通和

截止两种状态来分别表示 0 和 1 的。

从逻辑上讲，存储器是由一个个存储单元构成的，它也正是利用这一个个的存储单元来存储或记忆二进制信息的。存储器所有存储单元的总数称为存储器的存储容量，通常用单位 KB(千字节)、MB(兆字节)、GB(吉字节)等表示，存储容量越大，表示存储器能够存储记忆的信息量越大。

4. 输入输出接口与设备

输入输出系统由输入输出设备和与设备配套的适配器(Adaptor)或接口电路组成。

计算机的输入输出设备又称为外围设备，它们主要完成人机间的信息交换。人们通过输入设备将以某种形式表示的信息转换为计算机内部所能接收和识别的二进制数据表示，由计算机运算处理后，将产生的结果通过输出设备以人或其他计算机所能接收和识别的信息形式输出。现代计算机的标准输入设备包括键盘、鼠标等，标准输出设备有显示器等。除此之外，还有各种各样满足不同用途的输入输出设备，如打印机、绘图仪、扫描仪、光笔以及各种图像输入输出设备、语音输入输出设备等。

由于输入输出设备种类繁多，在处理速度、数据格式、机械及电器特性等方面差异较大，因此，CPU 与输入输出设备间的数据交换通常是通过相应输入输出适配器(又称输入输出接口)来实现的。例如，CPU 对显示器的输出控制就是通过显卡(又称显示器适配卡)来完成的。

除了上述各部件外，计算机系统中还必须有总线。系统总线是构成计算机系统的骨架，是多个系统部件之间进行数据传送的公共通路。借助系统总线，计算机在各系统部件之间实现传送地址、数据和控制信息的操作。

知识拓展

冯·诺依曼和非冯·诺依曼体系结构

在最早期的电子计算机中，编程就是利用各种导线进行接插连线。由于没有计算机分层的概念，对早期的计算机每进行一次编程都是一项非常大的布线工程。

在 ENIAC 计算机研制的同时，冯·诺依曼与莫尔小组也在合作研制 EDVAC 计算机，在 EDVAC 计算机中首次采用了存储程序的思想，这种采用存储程序的计算机就称为冯·诺依曼计算机。冯·诺依曼结构的计算机具有以下一些特点：

(1) 计算机由运算器、控制器、存储器、输入设备和输出设备五部分组成。

(2) 采用存储程序的方式，要执行的程序和数据事先存放在存储器中。

(3) 采用二进制编码表示数据。

(4) 程序是指令的集合，指令在存储器中按执行顺序存放。

虽然现代计算机普遍采用冯·诺依曼体系结构设计，但由于冯·诺依曼计算机中 CPU 与存储器之间单一通道这一瓶颈问题使得计算机性能的提高受到了限制，因此，人们不断努力，试图突破冯·诺依曼结构的限制，从而在历史上也曾出现过一些非冯·诺依曼结构计算机的研究或对冯·诺依曼结构的一些改进，如利用人脑模型的思想作为计算模式的神经网络计算机、利用生物学和 DNA 演化的思想开发的基因算法、量子计算和并行计算机等。在这些新的体系结构中，并行计算的概念是目前最成功和最流行的。

1.5　计算机语言

计算机语言是人与计算机进行交流的语言，通过计算机语言，人们可以告知或命令计算机需要做什么，要完成什么样的工作等。计算机语言主要用于提供人们编写计算机软件程序，因此，又称为程序设计语言。

计算机语言按照与硬件相关程度由高到低分为机器语言、汇编语言和高级语言。机器语言属于硬件机器级语言，是一种用二进制代码表示的能够被计算机硬件直接识别和执行的语言。不同计算机（主要是 CPU）的机器语言是不同的。由于机器语言不便于程序员掌握和使用，因此现在很少使用机器语言编制程序。

汇编语言是一种采用助记符表示的程序设计语言。汇编语言的指令和机器语言的指令在很大程度上是一一对应的，它不能被计算机的硬件直接识别，但却便于程序员记忆。用汇编语言编写的程序首先由一个系统软件——汇编程序翻译成机器语言的机器代码，然后在计算机上执行。汇编语言的语法、语义结构等和机器语言基本一致，同样与机器紧密相关。用汇编语言编写程序同样比较复杂，且所编制的程序代码难以移植。

高级语言是与机器无关的程序设计语言，它具有更强的表达能力，采用一种更接近自然的表达方式表示数据的运算和程序的控制结构等，使程序员容易学习掌握。目前的应用软件主要使用高级语言编制实现。随着 Windows 系统的推出和近些年多媒体技术的应用、发展，高级语言正向着面向对象、可视化的方向发展，使得程序设计的效率更高。

在各种层次的计算机语言中，机器语言是硬件直接识别的语言，其他各种语言可看成相应的虚拟机的机器语言。某一层次的虚拟机能够直接理解的语言称为该层的虚拟机语言。虚拟机语言层次越高，语言的表达能力就越强。处在某一级虚拟机的程序员只需知道这一级的语言及虚拟机属性，其下层的特性无须知道，即下层的特性对程序员来说是透明的。

用程序设计语言编写的程序称为源程序。高级语言的源程序可以通过两种方式转换成机器语言程序：一种是通过编译程序在运行之前将源程序转换成机器语言程序；另一种是通过解释程序进行解释执行。

1.6　计算机系统的分层组织结构

从一般使用者的角度来看，计算机系统是由硬件和软件组成的，而计算机软件根据其在计算机系统中所起的作用又可进一步分为系统软件和应用软件。系统软件是指能够对计算机硬件资源进行管理，对用户方便使用计算机硬件资源提供服务的软件，其核心就是操作系统。应用软件则是人们使用各种计算机语言为解决各种应用问题而编制的程序。因此，从这一层面上看，计算机系统自下而上可以看成由三个层次构成，即计算机硬件、系统软件和应用软件，下层为上层功能的实现提供支持。

而从计算机设计者的角度看，计算机系统可以进一步划分为不同的层次来实现其功能。这种划分可以看成概念上的划分，但却是十分有意义的。可以设想计算机是按照不同的层次结构来建造的。这里的每一个层次都实现某项特定的功能，并有一个特定的假想机器与

之对应。对应计算机的每一个层次的这种假想机器称为虚拟机。每一层的虚拟机都执行自己特有的指令集,必要时还可以调用较低层次的虚拟机来完成各种任务。如图 1-17 所示是一个业界普遍接受的代表不同抽象的虚拟机器的计算机组织结构层次图。

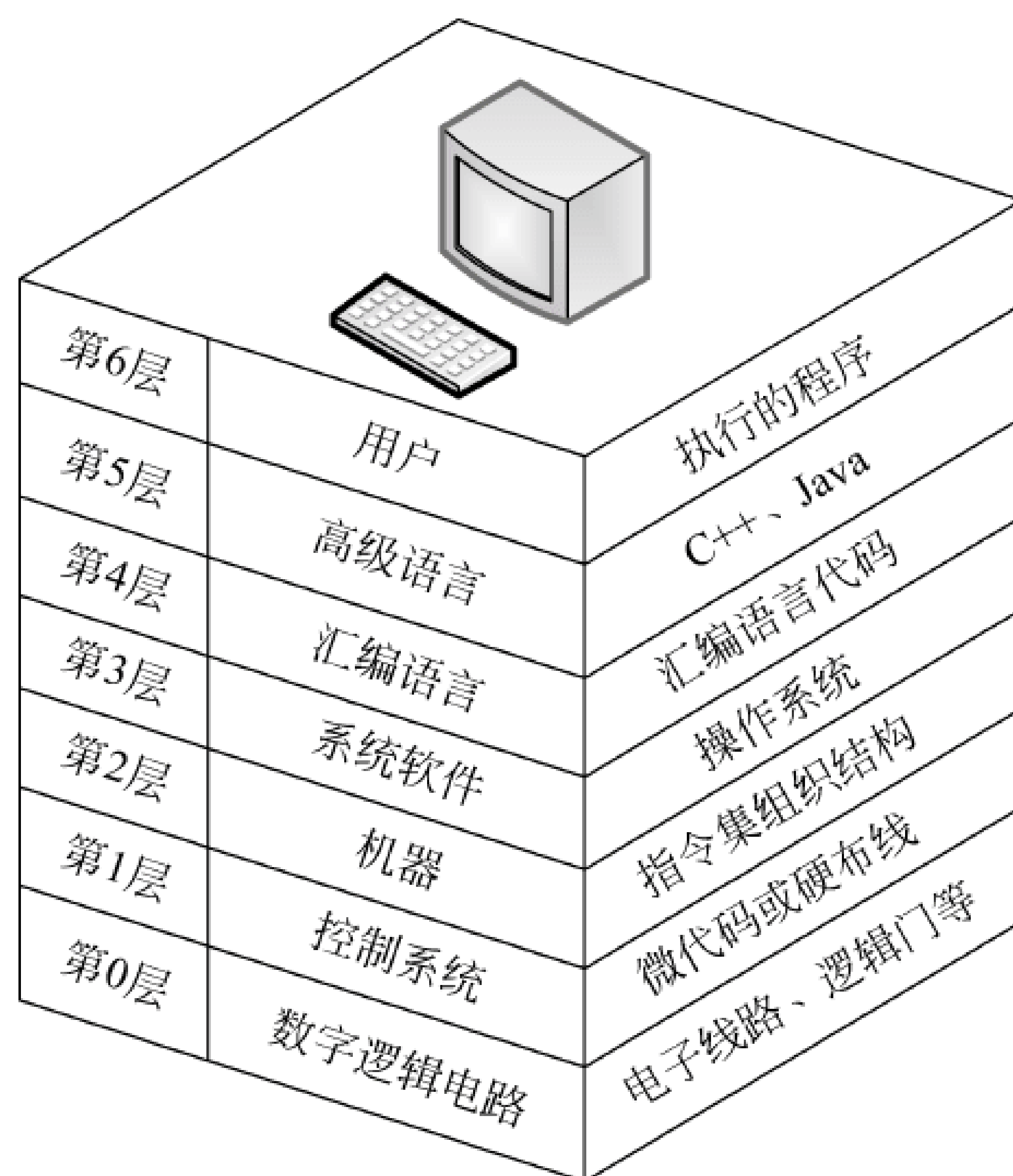


图 1-17 计算机系统的分层组织结构

第 6 层是用户层,也是面向一般用户的层次,换句话说,一般用户在使用计算机时所看见的就是这一层次。在这一层次上,用户可以运行各种应用程序,如字处理程序、制表程序、财务处理程序、游戏程序等。对用户层而言,其他各较低的层次可以是不可见的,也就是说,用户不必了解各底层是如何实现的。

第 5 层是高级语言层,它由各种高级语言组成,如 C、C++、Java、Web 编程语言等。这些高级语言提供该层用户为完成某一特定任务而编写高级语言程序。一方面,所编写的这些高级语言程序提供给上层用户层的用户使用,另一方面这些高级语言程序是通过编译或解释成低级语言来实现的。虽然使用这些高级语言编写程序代码的程序员需要了解所使用语言的语法、语义及各种语句等,但这些语法、语义的实现及语句的执行过程对他们来讲是透明的。

第 4 层是汇编语言层,它包括各种类型的汇编语言。每一台计算机都有自己的汇编语言,上层的高级语言首先被翻译成汇编语言,再进一步翻译成计算机直接识别的机器语言。计算机通过执行机器语言程序来最终完成用户所要求的功能。

第 3 层是系统软件层,其核心就是操作系统。操作系统对用户程序使用计算机的各种资源(CPU、存储器、输入输出设备等)进行管理和分配。例如,当某一用户程序需要运行时,首先由操作系统将其调入内存中,这其中需要操作系统为其分配内存空间进行存储。再如某程序需要使用某一输出设备进行结果的输出时,需要操作系统为其提供对该设备的控制等。

第 2 层是机器层,这是面向计算机体系结构设计者的层次。计算机系统设计者首先要确定计算机的体系结构,如计算机的硬件包含哪些部件,采用什么样的连接结构和实现技术

等。在这一层次上提供的是机器语言，也是机器唯一能直接识别的语言，其他各种语言的程序最终都必须翻译成机器语言程序，由计算机通过其硬件实现相应的功能。

第1层是控制层，这一层的核心是计算机硬件控制单元。控制单元会逐条接收来自上层的机器指令，然后分析译码，产生一系列的操作控制信号，并由这些控制信号控制下层的逻辑部件按照一定的时间顺序有序地工作。

第0层是数字逻辑层，在这里所面对的是计算机系统的物理构成：各种逻辑电路和连接线路，它们是组成计算机硬件的基础。

计算机系统的各个层次并不是孤立的，而是互相关联、互相协作的。一般来讲，下层为上层提供服务或执行上层所要求的功能，而上层通过使用下层提供的服务完成一定的功能。计算机这种层次划分的好处是：某一个层次的设计者可以专注于该层功能的实现，通过采用各种技术，提高各层次的性能，从而提高计算机系统整体的性能。

从功能上讲，任何可以利用软件实现的功能也可以利用硬件实现，反之，任何可以利用硬件实现的功能同样也可以利用软件实现，这就是所谓的硬件和软件等效原理。计算机中的任何操作可以由软件实现，也可以由硬件实现，某一台计算机功能是采用硬件实现还是采用软件实现，取决于计算机的性能价格比。当研制一台计算机的时候，设计者必须明确软硬件的功能划分。随着大规模集成电路和计算机体系结构的发展，由硬件实现的功能范围逐步扩大，这也就使计算机的处理速度越来越快，性能越来越高。

目前，计算机在机器语言级是由硬件直接实现，即由硬件实现对机器指令的解释和执行。随着发展，今后的计算机将由硬件直接实现更高级的功能，如通过硬件直接解释执行高级语言的语句而不需要先经过编译程序的处理，因此传统的软件部分今后完全有可能"固化"，即通过所谓的固件方式实现。

本章小结

本章主要讲述了以下内容：

(1) 计算机的发展历程。从机械计算机发展到电子计算机，电子计算机按所使用的电子元器件划分年代(电子管计算机、晶体管计算机、中小规模集成电路计算机和大规模超大规模集成电路计算机年代)。

(2) 计算机的种类。按照1989年由IEEE科学巨型机委员会提出的运算速度分类法，将计算机分为巨型机、大型机、小型机、工作站和微型计算机。根据计算机的综合性能指标，将计算机分为高性能计算机、微型计算机、工作站、服务器和嵌入式计算机。

(3) 计算机的性能指标。主要包括吞吐量、响应时间，CPU时钟周期、主频、CPI、CPU执行时间，MIPS、MFLOPS、GFLOPS、TFLOPS、PFLOPS等。

(4) 计算机的硬件组成。主要包括中央处理器(CPU)、存储器和输入输出系统，CPU主要由运算器和控制器组成，这也是冯·诺依曼提出的计算机组成结构。

(5) 计算机语言。由低到高分为机器语言、汇编语言和高级语言。

(6) 计算机的层次结构。从计算机的基本硬件开始分为数字逻辑层、控制层、机器层、系统软件层、汇编语言层、高级语言层和用户层。

习题一

一、名词解释

1. 集成电路　　2. 系列机　　3. 通用机　　4. 虚拟机

二、选择题

1. 世界上第一台电子计算机 ENIAC 中使用的基本元器件是(　　)。

A. 机械装置　　B. 电子管　　C. 晶体管　　D. 集成电路

2. 现代计算机中使用的基本元器件是(　　)。

A. 电子管　　B. 晶体管　　C. SSI 和 MSI　　D. LSI 和 VLSI

3. 按照 1989 年 IEEE 科学巨型机委员会提出的运算速度分类法,计算机种类中不包括(　　)。

A. 巨型机　　B. 小型机　　C. 小巨型机　　D. 微型机

4. 运算器中一般应包含(　　)。

A. ALU 和寄存器组　　B. ALU 和 IR

C. ALU 和 DR　　D. ALU 和 AR

5. 冯·诺依曼计算机工作方式的基本特点是(　　)。

A. 多指令流单数据流　　B. 多指令流多数据流

C. 堆栈操作　　D. 按地址访问并顺序执行指令

三、综合题

1. 什么是摩尔定律? 它对计算机集成电路技术的发展有何影响?
2. 解释 SSI、MSI、LSI 和 VLSI 的区别。
3. 计算机是如何进行分类的?
4. 计算机从硬件上讲包括哪些主要部件?
5. 阐述 CPU 中的运算器和控制器的基本结构。
6. 指令译码器是如何工作的?
7. 阐述冯·诺依曼体系结构的特点。
8. 计算机是如何划分层次的? 划分层次有何意义?
9. 什么是硬件和软件等效原理?
10. 查阅资料,简要概括一下我国在处理器芯片技术领域的发展情况。
11. 查阅资料,简要概括当前计算机集成电路技术在芯片集成度方面发展的情况。
12. 查阅资料,进一步说明计算机系统性能评测的方法和指标体系。
13. 衡量 CPU 性能指标的一个参数是 CPI(Cycles Per Instruction),查阅有关资料,了解 CPI 的含义。
14. 衡量 CPU 性能指标的一个参数是 MIPS(每秒百万条指令),查阅有关资料,了解 MIPS 的含义。
15. 假设您现在要为自己购置一台计算机,请说明您使用这台计算机的用途,并列出您所知道的现在计算机的性能配置情况(如 CPU、主板、内存等)。

第2章 数据的机器级表示及运算

早期的计算机主要用于科学计算，以帮助人们提高计算的速度，为此在计算机中必须使用一定的方法表示数值数据，并完成各种运算。紧接着，人们使用计算机来处理各种文字信息，以提高对文本处理的效率，为此在计算机中又必须使用一定的方法表示字符等非数值数据。随着计算机在各种应用领域的发展，计算机所能表示的信息种类越来越多，如声音、图形、图像、视频和动画等。但无论什么样的信息或数据，在计算机中都是通过二进制编码的方法来表示的。

本章首先介绍二进制等基本的进位记数制；再介绍在计算机中是如何对人们日常处理的数值数据和非数值数据（主要包括字符、汉字等）进行二进制编码表示的；然后介绍数值数据在计算机中的二进制运算方法和实现；最后介绍对计算机中的数据在传递过程中产生的差错进行检测而使用的数据校验码。

2.1 数制及转换

生活中的数值计算是采用十进制进行的，而在计算机中采用的却是二进制。另外，为了方便地表示二进制，人们又设计了其他进位记数制，如十六进制、八进制等。

2.1.1 进位记数制

将数字符号按序排列成数位，并遵照某种由低位到高位的进位方式记数来表示数值的方法，称为进位计数制，简称计数制。

无论使用哪种进位记数制，数值的表示都包含两个基本要素：基数和位权。

一种进位计数制允许使用的基本数字符号的个数称为这种进位计数制的基数。一般而言，K 进制数的基数为 K，可供选用的基本数字符号有 K 个，它们分别为 $0 \sim K-1$，每个数位计满 K 就向其高位进 1，即"逢 K 进 1"。

进位计数制中每位数字符号所表示的数值，等于该数字符号值乘以一个与数字符号所处位置有关的常数，这个常数就称为位权，简称权。位权的大小是以基数为底、数字符号所处位置的序号为指数的整数次幂。各数字符号所处位置的序号计法为：以小数点为基准，整数部分自右向左依次为 0、1、…递增，小数部分自左向右依次为 −1、−2、…递减。

任何进制数的值都可以表示成该进制数中各位数字符号值与相应位权乘积的累加形式，该形式称为按权展开的多项式和。一个 K 进制数 $(N)_K$，用按权展开的多项式和形式可表示为

$$(N)_K = D_m K^m + D_{m-1} K^{m-1} + \cdots + D_1 K^1 + D_0 K^0 + D_{-1} K^{-1} + D_{-2} K^{-2} + \cdots + D_{-n} K^{-n} \tag{2.1}$$

例如，十进制数$(123.4)_{10}$可表示为

$$(123.4)_{10} = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1}$$

任何进制数按权展开的多项式和的值，就是该进制数所对应的十进制数的值。换言之，任何进制与十进制数的转换只需求该进制数的按权展开的多项式和的值即可。

1. 二进制数

计算机内部使用的是二进制。所有的数值数据和非数值数据，都由二进制 0、1 这两个数字符号组合而成，它们统称为“二进制代码”。二进制的基数为 2，只有 0 和 1 两个数字符号，计数“逢 2 进 1”，用按权展开的多项式和形式可表示为

$$(N)_2 = D_m 2^m + 2^{m-1} D_{m-1} + \cdots + 2^1 D_1 + 2^0 D_0 + 2^{-1} D_{-1} + 2^{-2} D_{-2} + \cdots + 2^{-n} D_{-n} \tag{2.2}$$

例如，二进制数$(1011.1)_2$可表示为

$$(1011.1)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1}$$

计算机内部采用二进制表示，具有以下优点：

(1) 技术容易实现。实际上，一个二进制“位”0 或 1 在计算机中是通过电子器件的两种状态来表示的，如电子开关的“开”和“关”、三极管的“导通”和“截止”、电位的“高”和“低”等两种状态，而这正符合电子器件的状态特性。

(2) 运算规则简单。两个一位二进制数的加、减运算规则各仅有 4 种，如加法：

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{(向高位进 1)}$$

(3) 与逻辑量吻合。逻辑量 1、0 表示一个事物的正、反两个方面，如是/非、真/假、对/错等。

2. 十六进制数

二进制数位数多、数字符号单调，不便于书写和记忆。在计算机应用中，通常用与二进制数有简单对应关系的十六进制数作为数据的书写形式。

十六进制的基数为 16，由 0～9、A～F 共 16 个数字、字母符号组成。其中，0～9 共 10 个数字符号，含义与十进制数相同，A～F 共 6 个字母符号的值分别对应十进制数的 10～15，计数“逢 16 进 1”，用按权展开的多项式和形式可表示为

$$(N)_{16} = 16^m D_m + 16^{m-1} D_{m-1} + \cdots + 16^1 D_1 + 16^0 D_0 + 16^{-1} D_{-1} + 16^{-2} D_{-2} + \cdots + 16^{-n} D_{-n} \tag{2.3}$$

例如，十六进制数$(28A.C)_{16}$可表示为

$$(28A.C)_{16} = 2 \times 16^2 + 8 \times 16^1 + 10 \times 16^0 + 12 \times 16^{-1}$$

3. 八进制数

早期也常用八进制数作为计算机应用中数据的书写形式。

八进制的基数为8，由0～7共8个数字组成，计数"逢8进1"，用按权展开的多项式和形式可表示为

$$(N)_8 = 8^m D_m + 8^{m-1} D_{m-1} + \cdots + 8^1 D_1 + 8^0 D_0 + 8^{-1} D_{-1} + 8^{-2} D_{-2} + \cdots + 8^{-n} D_{-n} \quad (2.4)$$

例如，八进制数$(35.7)_8$可表示为

$$(35.7)_8 = 3 \times 8^1 + 5 \times 8^0 + 7 \times 8^{-1}$$

2.1.2　数制的转换

1. 二进制数、十六进制数转换成十进制数

如前所述，任何进制数只要求得其按权展开的多项式之和，该和值便是对应的十进制数。所以，二进制数或十六进制数转换为十进制数可以使用以下的"加权求和法"。

具体方法为：将要转换的二进制数或十六进制数表示成按权展开的多项式和的形式，然后逐项相加，所得的和值便是对应的十进制数。

【例2.1】　将二进制数$(1011.1)_2$转换成对应的十进制数。

解：$(1011.1)_2 = 1\times2^3+0\times2^2+1\times2^1+1\times2^0+1\times2^{-1}=11.5$

【例2.2】　将十六进制数$(3C.8)_{16}$转换成对应的十进制数。

解：$(3C.8)_{16}=3\times16^1+12\times16^0+8\times16^{-1}=60.5$

2. 十进制数转换成二进制数

十进制数转换成二进制数，相对二进制数转换成十进制数来要复杂一点。其中，十进制数的整数部分和小数部分要用不同的方法加以处理。

十进制数的整数部分采用"除2取余"法进行转换。

具体方法为：将要转换的十进制整数除以二进制的基数2，取商的余数作为二进制整数最低位的系数K_0，继续将商的整数部分除以2，再取商的余数作为二进制整数次低位的系数K_1，…，这样依次相除，直至商为0为止，最后一位余数作为二进制整数最高位的系数K_n。余数序列$K_nK_{n-1}\cdots K_1K_0$便构成了对应的二进制数。

【例2.3】　将十进制数43转换成对应的十进制数。

解：

	余数	二进制数位
2 | 43		
2 | 21	1	K_0
2 | 10	1	K_1
2 | 5	0	K_2
2 | 2	1	K_3
2 | 1	0	K_4
0	1	K_5

所以，$(43)_{10}=(101011)_2$。

十进制的小数部分采用“乘 2 取整法”进行转换。

具体方法为：将要转换的十进制小数部分乘以二进制的基数 2，取积的整数部分作为二进制小数的最高位的系数 K_{-1}，继续将积的小数部分乘以 2，再取积的整数部分作为二进制小数次高位的系数 K_{-2}，…，这样依次相乘，直至积的小数部分为 0 或达到所需精度为止，最后一位积的整数部分作为二进制小数最低位的系数 K_{-m}。积的整数部分序列 $0.K_{-1}K_{-2}\cdots K_{-m+1}K_{-m}$ 便构成了对应的二进制数。

【例 2.4】 将十进制数 0.6875 转换成对应的二进制数。

解：

0.6875	积的整数部分	二进制数位
× 2		
1.375	1	K_{-1}
0.375		
× 2		
0.75	0	K_{-2}
× 2		
1.5	1	K_{-3}
0.5		
× 2		
1	1	K_{-4}

所以，$(0.6875)_{10}=(0.1011)_2$。

进行数制转换时，整数部分经转换后必定仍为整数，小数部分经转换后也仍为小数。所以，对于既有整数部分，又有小数部分的十进制数，可按上述方法分别进行转换，然后组合在一起即为所转换的结果。

3. 二进制数与十六进制数的相互转换

十六进制数与二进制数存在着简单的转换关系，每 1 位十六进制数正好对应 4 位二进制数，它们的对应关系如表 2-1 所示。

表 2-1 十六进制数和二进制数的对应关系

十 六 进 制	二 进 制	十 六 进 制	二 进 制
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

由于 1 位十六进制数正好对应 4 位二进制数，所以二进制数转换成十六进制数时，只要以小数点为界，整数部分向左、小数部分向右分成 4 位一组，各组分别用对应的 1 位十六进制数表示，即可得到所求的十六进制数。两头的分组不足 4 位时，在小数点左边的高位和小

数点右边的低位可用 0 补足。

【例 2.5】 将二进制数 1011010.10111 转换成对应的十六进制数。

解：在二进制数 1011010.10111 小数点左边的高位补 1 个 0，小数点右边的低位补 3 个 0，则

$$1011010.10111 = \underset{5}{\underline{0101}}\ \underset{A}{\underline{1010}}.\underset{B}{\underline{1011}}\ \underset{8}{\underline{1000}}$$

所以，$(1011010.10111)_2 = (5A.B8)_{16}$。

同理，将十六进制数的每 1 位转换成对应的 4 位二进制数，便可实现十六进制数到二进制数的转换。

八进制数与二进制数之间的转换方法和十六进制数与二进制数之间的类似，所不同的只是 1 位八进制数对应 3 位二进制数。

至于十六进制与十进制数之间的转换，同样可用按权展开的多项式之和，以及整数部分用“除基取余”法、小数部分用“乘基取整”法来实现。不过，此时的基数为 16。当然，更简单实用的方法是，借二进制数作为过渡，用“十六↔二↔十”的转换方法来实现。

知识拓展

二进制的游戏

为了加深对二进制的印象，我们来做一个游戏。

拿出一张纸，假设这张纸足够大。将它对折，再对折，再对折，……，共对折 32 次，那么设想一下，对折后的纸会有多厚呢？

我们现在来计算一下，也许结果会出乎预料，让你大跌眼镜。

一张普通纸的厚度大约 0.04mm，也即 4×10^{-5}m。将纸对折一次后共有 2 张纸，再对折后共有 4(即 2^2)张纸，再对折后共有 8(即 2^3)张纸……对折 32 次后共有 2^{32}张纸。

$$2^{10} = 1024 \approx 1000 = 10^3$$
$$2^{20} \approx 10^6$$
$$2^{30} \approx 10^9$$
$$2^{32} \approx 4 \times 10^9$$

因此，经 32 次对折后的纸的厚度约为

$$(4 \times 10^{-5}) \times (4 \times 10^9)\text{m} = 1.6 \times 10^5\ \text{m}$$

即 1600km。

实际上，二、十进制的指数间存在以下对应关系：

$$10^n = 2^{30}$$
$$n = \lg 2^{30} = 30\lg 2 \approx 30 \times 0.3013 > 9$$

2.2 数值数据的机器表示

数值数据是一种有正负之分的带符号数，在日常生活中，通常用正号＋表示正数，用负号－表示负数。数值数据的符号连同其数值部分一起编码成二进制数，不同的编码形式产

生了不同的机器表示法。

2.2.1 数据的机器数表示

所谓数据的机器数表示是指计算机硬件能够直接表示、存储和处理的数据形式。数值数据是带符号数,即有正负之分。在计算机中,数的符号(+或-)和数的值一样都要采用二进制 0、1 编码。对数值数据的编码表示常用的有原码、补码、反码和移码表示等。这几种表示法都将数据的符号数码化。为了区分一般书写时表示的数和机器中编码表示的数,称前者为真值,后者为机器数。机器数包含两部分:符号位和数值部分。

1. 原码表示法

原码表示法是一种比较直观的表示方法,其具体表示方法是:符号位表示该数的符号,正(+)用"0"表示,负(-)用"1"表示,而数值部分保持与其真值相同。

设纯小数的原码形式为 $x=x_0.x_1x_2\cdots x_n$,则原码表示的定义为

$$[x]_{原} = \begin{cases} x & 1 > x \geqslant 0 \\ 1 - x = 1 + |x| & 0 \geqslant x > -1 \end{cases} \tag{2.5}$$

式中$[x]_{原}$是机器数,x 是真值。

【例 2.6】 $x=+0.1001$,则$[x]_{原}=0.1001$;

$x=-0.1001$,则$[x]_{原}=1.1001$。

设纯整数的原码形式为 $x=x_0x_1x_2\cdots x_n$,则原码表示的定义为

$$[x]_{原} = \begin{cases} x & 2^n > x \geqslant 0 \\ 2^n - x = 2^n + |x| & 0 \geqslant x > -2^n \end{cases} \tag{2.6}$$

原码表示法有两个特点:

(1) 零的表示有"+0"和"-0"之分,故有两种形式:

$$[+0]_{原}=0.000\cdots0$$
$$[-0]_{原}=1.000\cdots0$$

(2) 符号位 x_0 的取值为:当真值为正数时,x_0 为 0;当真值为负数时,x_0 为 1。

原码表示法的优点是比较直观、简单易懂,但它的最大缺点是加法运算复杂。这是因为,当两数相加时,如果是同号则数值相加;如果是异号,则要进行减法。而在进行减法运算时,还要比较绝对值的大小,然后减去小数,最后还要给结果选择恰当的符号。显然,利用原码作加减法运算是不太方便的。另外,原码的零是不唯一的。

2. 补码表示法

补码表示法是计算机中实际采用的一种编码方法,与原码表示相同的是其符号位表示该数的符号,正(+)用 0 表示,负(-)用 1 表示,但数值部分有所不同。

设纯小数的补码形式为 $x=x_0.x_1x_2\cdots x_n$,则补码表示的定义为

$$[x]_{补} = \begin{cases} x & 1 > x \geqslant 0 \\ 2 + x = 2 - |x| & 0 \geqslant x > -1 \end{cases} \tag{2.7}$$

对于 0,在补码表示情况下只有一种表示形式,即

$$[+0]_{补}=[-0]_{补}=0.000\cdots0$$

【例 2.7】 $x=+0.1001$,则$[x]_{补}=0.1001$;

$x=-0.1001$,则$[x]_{补}=1.0111$。

对于纯整数 $x=x_0x_1x_2\cdots x_n$,补码表示的定义是

$$[x]_{补}=\begin{cases}x & 2^n>x\geqslant0\\2^{n+1}+x=2^{n+1}-|x| & 0\geqslant x>-2^n\end{cases}\quad(2.8)$$

补码有两条重要的性质:

(1) 补码的零是唯一的。

(2) 补码的减法可以化为加法实现,即

$$\begin{aligned}[X+Y]_{补}&=[X]_{补}+[Y]_{补}\\[X-Y]_{补}&=[X]_{补}+[-Y]_{补}\end{aligned}\quad(2.9)$$

采用补码表示法进行加减法运算比原码更加方便。因为不论数是正还是负,机器总是做加法,减法运算可转换成加法运算实现。

但根据补码定义,正数的补码与原码形式相同,而求负数的补码要减去$[x]$。为了用加法代替减法,结果还得在求补码时做一次减法,这显然是不方便的。从下面介绍的反码表示法中可以获得求负数补码的简便方法,解决负数的求补问题。

3. 反码表示法

反码表示法中,符号的表示法与原码相同;而对于数值部分,正数的反码与正数的原码数值部分相同,负数的数值部分则通过将负数原码的数值部分各位取反(0 变 1,1 变 0)得到。

设纯小数的反码形式为 $x=x_0x_1x_2\cdots x_n$,则反码表示的定义为

$$[x]_{反}=\begin{cases}x & 1>x\geqslant0\\2-2^{-n}+x=2-|x| & 0\geqslant x>-1\end{cases}\quad(2.10)$$

【例 2.8】 $x=+0.1001$,则$[x]_{反}=0.1001$;

$x=-0.1001$,则$[x]_{反}=1.0110$。

对于 0,在反码情况下有两种表示形式,即

$$[+0]_{反}=0.000\cdots0$$
$$[-0]_{反}=1.111\cdots1$$

对于纯整数 $x=x_0x_1x_2\cdots x_n$,反码表示的定义是

$$[x]_{反}=\begin{cases}x & 2^n>x\geqslant0\\2^{n+1}-1+x & 0\geqslant x>-2^n\end{cases}\quad(2.11)$$

通过比较小数与整数的反码与补码的公式可得到

$$[x]_{补}=[x]_{反}+2^{-n}\qquad 0>x\geqslant-1$$
$$[x]_{补}=[x]_{反}+1\qquad 0>x\geqslant-2^{-n}$$

这就是通过反码求补码的重要公式。这两个公式告诉我们,若要将一个负数用补码表示,其方法是:符号位置 1,数值部分各位变反,末位加 1。

4. 移码表示法

移码主要用于表示后面要讲到的浮点数的阶码，而且通常表示的是纯整数。

对于纯整数 $x=x_0x_1x_2\cdots x_n$，移码表示的定义为

$$[x]_{移}=2^n+x \qquad 2^n>x\geqslant-2^n \tag{2.12}$$

【例 2.9】 $x=+1001$，则$[x]_{移}=11001$；

$x=-1001$，则$[x]_{移}=00111$。

通过对比补码和移码发现，对任意一个真值，其补码和移码的机器数表示中，数值部分相同，而符号恰恰相反。其实，用移码表示的机器数，当真值为正数时，其符号位为 1，当真值为负数时，其符号位为 0。

2.2.2 定点数和浮点数

日常表示的数据类型主要有两种：一是一般的数据表示形式，如 125、98.6 等；二是科学计数法表示的数据形式，如 1.25×10^8 等。这两种数据类型对应在计算机中的表示形式就是定点数和浮点数。

1. 定点数的表示方法

作为一个一般的十进制数据，在计算机中除了要表示其数值外，还要表示其符号（正或负）和小数点。符号可以使用一位二进制数表示，如“0”表示正号，“1”表示负号。而对于小数点则需要采取一些特殊的处理方法。

所谓定点数是指数据的小数点位置是固定不变的。由于定点数的小数点位置是固定的，因此小数点“.”就不需要表示出来了。在计算机中，定点数主要分为两种：一是定点整数，即纯整数；二是定点小数，即纯小数。

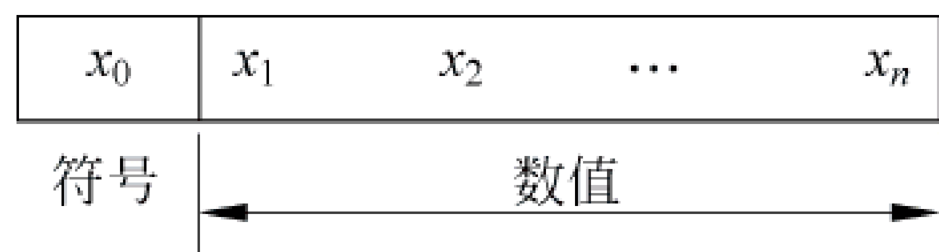


图 2-1 定点数表示

假设用一个 $n+1$ 位二进制来表示一个定点数 x，其中一位 x_0 用来表示数的符号位，其余 n 位数代表它的数值。这样，对于任意定点数 $x=x_0x_1x_2\cdots x_n$，其在机器中的定点数表示如图 2-1 所示。

如果数 x 表示的是纯小数，那么小数点位于 x_0 和 x_1 之间，其数值范围为

$$0\leqslant|x|\leqslant1-2^{-n}$$

如果数 x 表示的是纯整数，那么小数点位于最低位 x_n 的右边，其数值范围为

$$0\leqslant|x|\leqslant2^n-1$$

在采用定点数表示的机器中，对于非纯整数或纯小数的数据在处理前必须先通过合适的比例因子转换成相应的纯整数或纯小数，运算结果再按比例转换回去。目前计算机中多采用定点纯整数表示，因此将定点数表示的运算简称为整数运算。

2. 浮点数的表示方法

使用定点纯整数或纯小数表示数据，其优点是表示方法简单，处理容易。但由于在计算

机中表示数据的二进制位数(称为字长)是有限的,因此定点数所表示的数据范围也是很有限的,对于一些很大的数据就无法表示。例如,使用16位二进制表示纯整数,其表示范围仅为:0～16383(正数)。为此,人们吸取生活中十进制数据的科学计数法的思想,采用一种称为浮点数的表示法来表示更大的数。

在浮点数表示中,数据被分为两部分:尾数和阶码。尾数表示数的有效数位,阶码则表示小数点的位置。加上符号位,浮点数据可以表示为

$$N = (-1)^S M R^E \tag{2.13}$$

其中,M(Mantissa)是浮点数的尾数,R(Radix)是基数,E(Exponent)是阶码,S(Sign)是浮点数的符号位,在计算机中表示如图2-2所示。

S	E_0	E_1　E_2　…　E_m	M_1　M_2　…　M_n
数符	阶符	阶码	尾数

图2-2　浮点数表示

在计算机中,基数R取2,是个常数,在系统中是约定的,不需要表示出来;阶码E用定点整数表示,它的位数越长,浮点数所能表示的数的范围越大;尾数M用定点小数表示,它的位数越长,浮点数所能表示的数的精度越高。

3. 浮点数的IEEE 754标准

早期的机器很多都支持浮点数表示,然而各自采用自己定义的表示形式,没有一个统一的标准,这给不同机器之间程序的移植带来了一定的困难。

1985年Intel打算为其8086微处理器配置一种浮点运算协处理器8087FPU。Intel公司请来了加州大学伯克利分校的William Kahan为其设计浮点数格式,Kahan又找来了两位专家协助他,于是就有了KCS组合(Kahn,Coonan and Stone)。他们共同完成了Intel的浮点数格式设计,而且完成得非常出色,以至于IEEE决定采用一个非常接近KCS的方案作为IEEE的标准浮点格式,也即IEEE 754标准。目前,几乎所有支持浮点运算的计算机都采用该标准,这也为程序的移植带来了方便。

IEEE 754标准从逻辑上采用上述的三元组$\{S,E,M\}$表示一个浮点数N,如图2-3所示。

N的实际值n由下列式子表示:

$$n = (-1)^s m 2^e \tag{2.14}$$

其中n、s、e、m分别为N、S、E、M对应的实际数值。

IEEE 754标准规定了三种浮点数格式:单精度、双精度、扩展精度。前两者正好对应C语言里的float、double或者FORTRAN里的real、double精度类型。

单精度浮点数N共32位,其中S占1位,E占8位,M占23位,如图2-4所示。

S	E	M

图2-3　三元组表示

31	30　　23	22　　0
S	E	M

图2-4　单精度浮点数

双精度浮点数 N 共 64 位，其中 S 占 1 位，E 占 11 位，M 占 52 位，如图 2-5 所示。

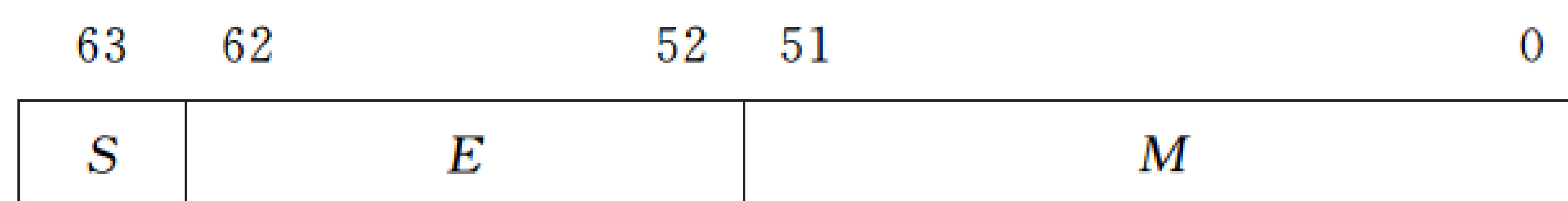


图 2-5 双精度浮点数

值得注意的是，M 虽然是 23 位或者 52 位，但它们只是表示小数点之后的二进制位数，也就是说，假定 M 为"010110011…"，在二进制数值上其实是".010110011…"。而事实上，标准规定小数点左边还有一个隐含位 1，也就是说 $m=1.M$，这一位并不需要存储，却可以提高一位尾数的精度。

对于浮点数而言，它还有一个特点，就是：一个数的浮点数表示可以不是唯一的。例如十进制数 123 可以表示成 1.1111011×2^7、0.11111011×2^8、0.011111011×2^9 等多种浮点数形式。为了保证一个数的浮点表示在机器中是唯一的，IEEE 754 标准规定：当 E 的二进制位不全为 0，也不全为 1 时，其尾数 m 必须是上述 $1.M$ 的表示形式，这称为浮点数的规格化表示。

此时 e 是采用移码表示的整数，e 值计算公式如下：

$$e = E - \text{bias},\quad \text{bias} = 2^{k-1} - 1 \tag{2.15}$$

其中，bias 为偏移值，k 为 E 的位数，对单精度浮点数来说，$k=8$，则 bias$=127$，即 $e=E-127$；对双精度浮点数来说，$k=11$，则 bias$=1023$，即 $e=E-1023$。

当一个浮点数的尾数为 0，不论其阶码为何值，或者当阶码的值遇到比它能表示的最小值还小时，不管其尾数为何值，计算机都把该浮点数看成零值，称为机器零。

当阶码 E 为全 0 且尾数 M 也为全 0 时，表示的真值为零，结合符号位 S 为 0 或 1，有正零和负零之分。当阶码 E 为全 1 且尾数 M 也为全 0 时，表示的真值为无穷大，结合符号位 S 为 0 或 1，有正无穷和负无穷之分。这样在单精度浮点数表示中，要除去 E 用全 0 和全 1 表示零和无穷大的特殊情况，指数的偏移值不选 128 而选 127。对于规格化浮点数，E 的范围变为 1～254，真正的指数值 e 则为 -126～$+127$。同样，在双精度位浮点数表示中，要除去 E 用全 0 和全 1 表示零和无穷大的特殊情况，指数的偏移值不选 1024，而选 1023。对于规格化浮点数，E 的范围变为 1～2046，真正的指数值 e 则为 -1022～$+1023$。

【例 2.10】 将十进制数 43.6875 转换成 IEEE 754 标准的单精度浮点数的二进制存储格式。

解：将十进制数 43.6875 转换成二进制数得

$$(43.6875)_{10}=(101011.1011)_2$$

再转换成浮点表示形式，其中尾数为 $1.M$ 的形式

$$101011.1011=1.0101111011\times2^5,\quad e=5$$

于是得到

$$S=0,\quad E=5+127=132,\quad M=010111011$$

最后得到十进制数 43.6875 的单精度浮点数的二进制存储格式为

0 10000100 01011101100000000000000

【例 2.11】 若浮点数 x 的 IEEE 754 标准存储格式为 $(41360000)_{16}$，求其浮点数的十进制值。

解：将$(41360000)_{16}$展开为二进制形式是

$$0\ 10000010\ 01101100000000000000000$$

符号位　　$S=0$

指数　　$e=E-127=10000010-01111111=00000011=(3)_{10}$

尾数　　$m=1.M=1.01101100000000000000000=1.011011$

于是有

$$x=(-1)^s\times m\times 2^e=+1.011011\times 2^3=+1011.011=(11.375)_{10}$$

知识拓展

位、字节、字的概念

位：又称二进制位(b)，其英文单词为 bit，是二进制数 binary digit 的缩写。"位"是计算机中最小的数据单位，每一位的状态只有两个，即 0 或 1。

字节(B)：英文单词为 Byte，1964 年，IBM System/360 大型机的设计者建立了一套以 8 个位为一组作为计算机存储器编址的基本单元的约定，他们称这种 8 个位的组合为一个字节。在机器中，一个字节常常可以被分为两半，每半 4 位，称为半字节(nibble)，高 4 位称为高半字节，低 4 位称为低半字节。1 个字节可以储存 1 个英文字母或者半个汉字，换句话说，1 个汉字占据 2 个字节的存储空间。现代计算机的编址单位就是字节，例如，我们注意到，厂家在列出机器的配置时，其中有一项是机器的内存，往往用多少 MB 表示，这里的 B 指的就是 Byte 字节的意思。

字：英文单词为 Word，一个字由若干个相邻的字节组成，字的位数称为字长。字长通常是 8 的整数倍，如 16 位、32 位、64 位等。字长对一台机器来说具有特殊的意义，它往往代表了该机器进行数据处理和运算的能力。一般来说，若某机器(主要指 CPU)为 64 位，一方面指该机器 CPU 的数据总线为 64 位，另一方面指该机器 CPU 内部的结构为 64 位，这里包括其内部的数据通路、运算部件及寄存器等。

另外，在计算机中常常会涉及一些计量单位如下：

$1K=2^{10}$

$1M=2^{20}$

$1G=2^{30}$

$1T=2^{40}$

2.3　非数值数据的机器表示

非数值数据主要是指如字符(串)、十进制数串、汉字、图形、图像等不计其数值大小的数据类型，对这些数据类型，在计算机中同样要采用一定的编码方式进行表示。

2.3.1　二进制编码的十进制数

现代计算机除了能将人们日常生活中的十进制数转换为上述二进制形式表示和处理外，还能直接使用二进制编码的方式表示十进制数位，常用的表示方法就是二进制编码的十

进制数(Binary-Coded Decimal),简称 BCD 码。

BCD 码是将一个十进制数的每个十进制数字编码成一个 4 位的二进制数,并且使用了 3 个 4 位二进制编码表示符号,其中 1111 表示无符号数,1100 表示正数,而 1101 则表示负数。十进制数的 BCD 编码方式如表 2-2 所示。

表 2-2 BCD 码编码

十进制数字	BCD 编码	十进制数字	BCD 编码
0	0000	7	0111
1	0001	8	1000
2	0010	9	1001
3	0011	正数	1100
4	0100	负数	1101
5	0101	无符号数	1111
6	0110		

从表 2-2 中可以看出,还有 6 个二进制编码(1010~1111)没有用于数字编码,虽然看起来几乎有 40%的资源浪费,但是在表示精度方面却得到了很大的提高。例如,十进制数 0.3 在按 2.2 节讲的方法转换为二进制时是一个循环小数 0.0100110011001…,截取 8 位存储则为 0.01001100,如果再从二进制转换回十进制却为 0.296 875,结果造成了一个约 1.05%的误差。而若采用 BCD 码表示,十进制数 0.3 可直接存储为 11110011(这里假定数据格式中隐含了小数点),根本就没有误差。

因为一个十进制数字的 BCD 编码只占一个 4 位的空间,所以在存储一个十进制数串时可以采取一种称为压缩十进制数串形式表示,具体方法:十进制数串的每个数字的 BCD 码连续存储,最后 4 位存储符号。

【例 2.12】 利用压缩的 BCD 编码存储-1265。

解:-1265 的 BCD 编码为

0001 0010 0110 0101

使用三个字节存储,其中在最低位数字后面加上符号,在高位补 0,如图 2-6 所示。

0000 0001	0010 0110	0101 1101

图 2-6 例 2.12

2.3.2 字符编码

现代计算机不仅要求能处理数值数据,还要求能处理非数值数据。非数值数据最常见的就是字符信息,如英文字母、标点符号、十进制数位以及诸如 $、%,+等符号和一些控制信息。在计算机中对于这一类数据的表示,同样采用的是二进制编码的方法。国际上出现了一些不同的编码方法,并在不同的领域或场合得以应用,以下列举几种常见的编码方法。

1. EBCDIC 码

IBM 公司在开发 IBM System360 计算机之前,使用了一种 6 位的扩展 BCD 码来表示

字符和数字。但是，这种编码方式所表示的字符很有限，例如，小写字符就无法表示。因此，IBM System360 系统的设计者进一步将 BCD 码扩展为 8 位，使用了一种扩展的二-十进制编码的交换码（Extended Binary Coded Decimal Interchange Code），简称 EBCDIC。表 2-3 按照区位-数字的形式给出了 EBCDIC 代码。字符表示采用在区位后面添加数字位的方法。例如，字符 a 是 1000 0001，数字 3 是 1111 0011。大写字母和小写字母的区别仅仅在第 2 位不同，只需简单地进行 1 位数字的翻位，即可实现大小写字母之间的转换。

表 2-3　EBCDIC 编码

区位＼数字	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	NUL	SCH	STX	ETX	PF	HT	LC	DEL		RLF	SMM	VT	FF	CR	SR	SI
0001	DLE	DC1	DC2	TM	RES	NL	BS	IL	CAN	EM	CC	CU1	IFS	IGS	IRS	IUS
0010	DS	SOS	FS		BYP	LF	ETB	ESC			SM	CU2		ENQ	ACK	BEL
0011			SYN		PN	RS	UC	EQT				CU3	DC4	NAK		SUB
0100	SP										[	。	<	(	+	!
0101	&										]	$	*	)	;	^
0110	—	/									|	,	%	—	>	?
0111	—										:	#	@	'	=	"
1000		a	b	c	d	e	f	g	h	i						
1001		j	k	l	m	n	o	p	q	r						
1010		—	s	t	u	v	w	x	y	z						
1011																
1100	{	A	B	C	D	E	F	G	H	I						
1101	}	J	K	L	M	N	O	P	Q	R						
1110	\		S	T	U	V	W	X	Y	Z						
1111	0	1	2	3	4	5	6	7	8	9						

2. ASCII 码

目前国际上普遍采用的字符系统是 7 位的美国国家信息交换标准字符码（American Standard Code for Information Interchang，ASCII 码）。ASCII 码是从使用了几十年的电传打字设备的编码方案中直接衍生出来的，这些电传设备使用 19 世纪 80 年代发明的 5 位摩尔编码。到了 20 世纪 60 年代，这种 5 位编码方式的局限性已经变得非常明显，国际标准化组织（ISO）就设计了一种 7 位的编码方案，并于 1967 年正式公布，这就是现在所称的 ASCII 码。

ASCII 定义了 10 个十进制数字、52 个英文字母（大小写）、32 个控制字符和 32 个特殊符号（如 $、%、+、= 等）等，共 128 个元素，另加一位奇偶校验位，共 8 位表示一个符号。表 2-4 列出了 7 位的 ASCII 码字符编码表。

ASCII 码规定 8 个二进制位的最高一位为 0，余下的 7 位可以给出 128 个编码，表示 128 个不同的字符。其中 95 个编码对应着计算机终端能敲入并且可以显示的 95 个字符，打印机设备也能打印这 95 个字符，包括大小写各 26 个英文字母、0～9 这 10 个数字符号、通用的运算符和标点符号+、—、*、/、>、=、< 等。

表 2-4　ASCII 字符编码表

$b_6\ b_5\ b_4$ / $b_3\ b_2\ b_1\ b_0$	000	001	010	011	100	101	110	111
0000	NUL	DEL	SP	0	@	P		p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	DEL	ETB		7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	$\vert$
1101	CR	GS	—	=	M	]	m	}
1110	SO	RS	.	>	N		n	~
1111	SI	US	/	?	O	_	o	DEL

另外的 33 个字符，其编码值为 0～31 和 127，则不对应任何一个可以显示或打印的实际字符，它们被用作控制码，控制计算机某些外围设备的工作特性和某些计算机软件的运行情况。

随着计算机硬件性能变得越来越可靠，对奇偶校验位的需求开始减弱。在 20 世纪 80 年代，微型计算机和外围设备的制造商开始利用奇偶校验位来提供扩展的字符集。

字符串是指连续的一串字符，通常情况下，它们占用主存中连续的多个字节，每个字节存储一个字符。当主存字由 2 个或 4 个字节组成时，在同一个主存字中既可按从低位字节向高位字节的顺序存放字符串的内容，也可按从高位字节向低位字节的顺序存放字符串的内容。这两种存放方式都是常用方式，不同的计算机可以选用其中任何一种。

3. 统一字符编码标准

无论是 EBCDIC 码还是 ASCII 码，都是建立在拉丁语系字母的基础上的。因此，这些编码方式在对非拉丁字母提供数据表示的能力方面受到了很大限制，而全球大多数人都使用非拉丁语系。随着全世界所有国家都开始使用计算机，每个国家都在设计出一套能够有效地表示自己本国语言的编码。然而，这些编码都不能与其他国家的编码相互兼容，这样也就在融入全球经济的道路上设置了另一种障碍。

为了防止这种情况继续恶化，1991 年成立了一个由工业和社会领导人组成的协会，并建立了一种新的国际信息交换代码，称为统一字符编码(Unicode)。

Unicode 是一个 16 位编码的字符表，可以向下兼容 ASCII 码和拉丁-1 字符集，并且与 ISO/IEC10646-1 国际字母表相一致。因为 Unicode 的基础是 16 位编码，所以能够对全世界所使用的每一种语言的大多数字符进行编码。如果这 16 位编码仍不够用，Unicode 还定

义了一种扩展机制,允许人们对更多的字符进行编码。

Unicode 的编码空间由五部分组成,如表 2-5 所示。

表 2-5 统一字符编码空间

字符类型	字符集说明	字符数目	十六进制数值
字母表	拉丁字母、希腊字母等	8192	0000—1FFF
符号	特殊符号、数学符号等	4096	2000—2FFF
CJK	中文、日文、韩文 语音符号和标点符号等	4096	3000—3FFF
Han	统一的中文、日文和韩文	40960	4000—DFFF
	Han 的扩展	4096	E000—EFFF
用户定义		4095	F000—FFFE

Unicode 有两套标准,一套是 UCS-2(Unicode-16),用 2 个字节为字符编码;另一套是 UCS-4(Unicode-32),用 4 个字节为字符编码。以目前常用的 UCS-2 为例,它可以表示的字符数为 $2^{16}=65\ 535$,基本上可以容纳所有的欧美字符和绝大部分的亚洲字符。在 Unicode 里,所有的字符被一视同仁。

目前,统一字符编码已成为美式计算机系统唯一使用的字符编码,其他大多数计算机制造商也都在一定程度上支持统一字符编码。

2.3.3 汉字的表示方法

普通的计算机键盘上主要分布的是 0～9 数字键、A～Z 字母键和一些功能键及控制键等。对数字和字母等符号的编码,计算机采用的是国际标准:ASCII 码。从键盘输入的值首先以键盘扫描码的形式被主机接收,然后由操作系统中的键盘中断处理程序转换为相对应的值的 ASCII 码。

但是在键盘上无法直接布置下成千上万的汉字,要让计算机能对汉字进行处理,必须解决两个问题,一是汉字的编码问题,二是汉字的输入问题。

1. 汉字的输入方法

为了能直接使用普通标准键盘把汉字输入计算机,就必须首先为汉字设计相应的输入编码方法,即一个汉字对应的键盘上的键的组合。当前采用的汉字输入编码方法主要有以下三类。

数字编码:常用的是国标区位码,即用数字串代表一个汉字输入码。区位码是将国家标准局公布的 6763 个两级汉字分为 94 个区,每个区分 94 位,实际上把汉字表示成二维数组,每个汉字在数组中的下标就是区位码。区码和位码各两位十进制数字,因此输入一个汉字需按键 4 次。例如,“中”字位于第 54 区 48 位,区位码为 5448。

数字编码输入的优点是无重码,且输入码与内部编码的转换方便,缺点是代码难以记忆。

拼音码:拼音码是以汉语拼音为基础的输入方法。凡掌握汉语拼音的人,不需要训练记忆,即可使用。但汉字同音字太多,输入重码率很高,因此按拼音输入后还必须进行同音

字选择,影响了输入速度。

字形编码:字形编码是用汉字的形状来进行的编码。汉字数虽多,但都是由一笔一画组成的,而组成汉字的部件和笔画是有限的。把汉字的笔画部件用字母或数字进行编码,按笔画的顺序依次输入,就能表示一个汉字。例如五笔字型编码是最有影响的一种字形编码方法,这种编码方法重码率较低,因此,一旦学习掌握,则输入速度较快。

除了上述三种编码方法之外,为了加快输入速度,在上述方法的基础上又进一步发展了词组输入、联想输入等多种快速输入方法。但它们都是通过键盘进行输入的。今后理想的输入方式是利用语音或图像识别技术,使计算机能认识汉字,听懂汉语,并将其自动转换为机内代码表示。目前这种理想已经成为现实,只是识别率和识别速度有待提高。

2. 汉字编码

汉字输入编码是为方便人们通过键盘进行汉字的输入而设计的编码,目前,这种编码不下几十种,也没有一个统一的标准。但无论采用哪种编码方法进行汉字的输入,在计算机中必须采用一种唯一的表示法来存储和处理汉字。

目前计算机处理汉字所用的编码标准是我国于 1980 年颁布的国家标准 GB 2312—80,即《中华人民共和国国家标准信息交换汉字编码》,简称国标码。国标码的主要用途是作为汉字信息交换码使用。GB 2312—80 一共收录了 7445 个字符,包括 6763 个汉字和 682 个其他符号。汉字区的内码范围高字节从 B0~F7,低字节从 A1~FE,占用的码位是 72×94=6768。其中有 5 个空位是 D7FA~D7FE。GB 2312—80 支持的汉字太少,1995 年的汉字扩展规范 GBK 1.0 收录了 21 886 个符号,它分为汉字区和图形符号区。汉字区包括 21 003 个字符。2000 年的 GB 18030 是取代 GBK 1.0 的正式国家标准。该标准收录了 27 484 个汉字,同时还收录了藏文、蒙文、维吾尔文等主要的少数民族文字。现在的 PC 平台必须支持 GB 18030,对嵌入式产品暂不作要求,手机、MP3 一般只支持 GB 2312—80。

国标码与 ASCII 码属同一制式,可以认为它是扩展的 ASCII 码。在 7 位 ASCII 码中可以表示 128 个信息,其中字符代码有 94 个。国标码是以 94 个字符代码为基础,其中任何两个代码组成一个汉字交换码,即由两个字节表示一个汉字字符。第一个字节称为“区”,第二个字节称为“位”。这样,该字符集共有 94 个区,每个区有 94 个位,最多可以组成 94×94=8836 个汉字。

在国标码表中,共收录了一、二级汉字和图形符号 7445 个。其中图形符号 682 个,分布在 1~15 区;一级汉字(常用汉字)3755 个,按汉语拼音字母顺序排列,分布在 16~55 区;二级汉字(不常用汉字)3008 个,按偏旁部首排列,分布在 56~87 区;88 区以后为空白区,以待扩展。

3. 汉字字模码

当要将汉字在显示器上显示或在打印机上打印输出时,是将汉字作为一种图形元素来对待的。因为对于显示器或打印机等输出设备来说,所有输出的内容,无论是文本还是图形、图像,都是通过点阵的方式来形成的,汉字也不例外。

对要输出的汉字,计算机是采用汉字字模码来表示的。汉字字模码是用点阵表示的汉字字形代码,它是汉字的输出形式。

根据汉字输出的要求不同，点阵的多少也不同。简易型汉字为 16×16 点阵，提高型汉字为 24×24 点阵、32×32 点阵，甚至更高。字模点阵的信息量很大，所占存储空间也很大。以 16×16 点阵为例，每个汉字都要占用 32 个字节，国标两级汉字要占用 256KB。因此字模点阵只能用来构成汉字库，不能用于机内存储。字库中存储了每个汉字的点阵代码。当显示输出或打印输出时才检索字库，输出字模点阵，得到字形。图 2-7 表示出了“王”字的点阵及代码。

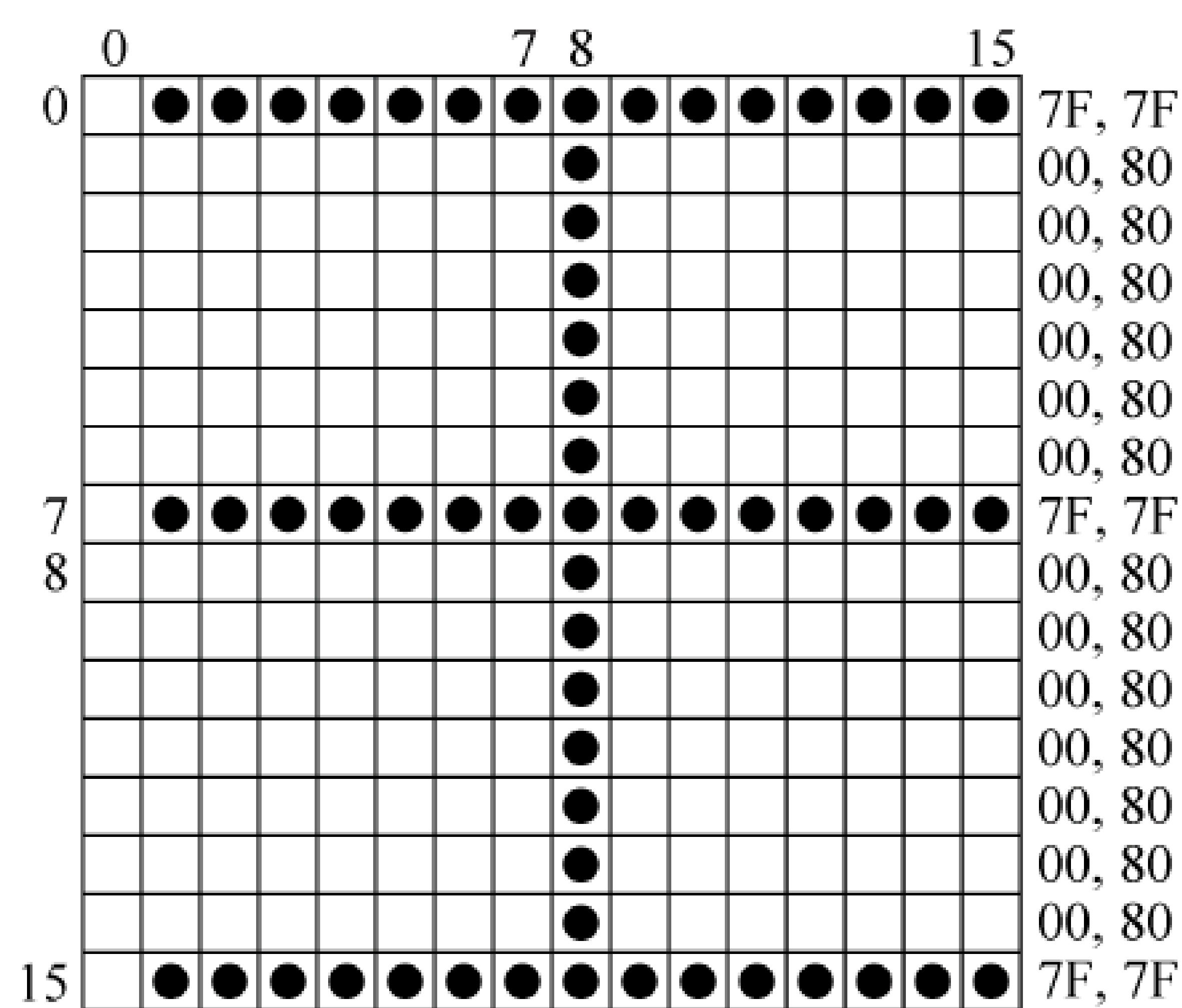


图 2-7　汉字点阵及编码

应注意的是，汉字的输入编码、汉字编码和汉字字模码是计算机中用于输入、存储和输出三种不同用途的编码，不要混为一谈。

2.4　定点数的运算及实现

定点数的运算主要包括定点数的加、减、乘、除等运算。从本节的讲述将会看到，二进制的减、乘、除等运算实际上是通过加和移位运算来实现的。另外，不同运算的算法会带来不同的运算及实现的复杂度。

2.4.1　定点数的加减运算

在机器中定点数是以补码形式表示的。因此，这里主要介绍补码的加减运算。

1. 补码的加法运算

根据补码的性质，可以得出以下结论：

(1) 用补码表示的两个数相加，其结果仍为补码。

(2) $[x+y]_{补}=[x]_{补}+[y]_{补}$。

(3) 符号位与数值位一同参加运算。

下面以纯小数的补码加法运算为例加以说明。

【例 2.13】　两个相加的数均为正数，设 $x=0.1001$，$y=0.0110$，则有

$$[x+y]_{补}=[0.1001+0.0110]_{补}=[0.1111]_{补}=0.1111$$

$$[x]_{补}+[y]_{补}=[0.1001]_{补}+[0.0110]_{补}=0.1001+0.0110=0.1111$$

即

$$[x+y]_{补}=[x]_{补}+[y]_{补}$$

【例 2.14】　两个相加的数一正一负，设 $x=0.1001$，$y=-0.0110$，则有

$$[x+y]_{补}=[0.1001+(-0.0110)]_{补}=[0.0011]_{补}=0.0011$$

$$[x]_{补}+[y]_{补}=[0.1001]_{补}+[-0.0110]_{补}=0.1001+1.1010=0.0011$$

同样

$$[x+y]_{补}=[x]_{补}+[y]_{补}$$

【例 2.15】 两个相加的数均为负数，设 $x=-0.1001, y=-0.0110$，则有

$$[x+y]_{补}=[(-0.1001)+(-0.0110)]_{补}=[-0.1111]_{补}=1.0001$$

$$[x]_{补}+[y]_{补}=[-0.1001]_{补}+[-0.0110]_{补}=1.0111+1.1010=1.0001$$

同样满足关系

$$[x+y]_{补}=[x]_{补}+[y]_{补}$$

2. 补码的减法运算

根据补码的性质，还可以得出以下结论：

（1）用补码表示的两个数相减，其结果仍为补码。

（2）$[x-y]_{补}=[x]_{补}+[-y]_{补}$。

（3）符号位与数值位一同参加运算。

【例 2.16】 设 $x=0.1001, y=0.0110$，则有

$$[x-y]_{补}=[0.1001-0.0110]_{补}=[0.0011]_{补}=0.0011$$

$$[x]_{补}+[-y]_{补}=[0.1001]_{补}+[-0.0110]_{补}=0.1001+1.1010=0.0011$$

从上述结论(2)可以看出，由于补码的减法运算可以转换为加法实现，所以计算时可以先求出 $-y$ 的补码，再做加法运算，这样在计算机中实现时，只需一个加法器即可完成加、减法运算，这也是计算机中使用补码表示的主要原因之一。

3. 二进制补码加法器的实现

补码的二进制加法是通过逐位加及进位实现的。加法器的基本电路是实现一位加的全加器，如图 2-8 所示。全加器的输入有三个：两个相加数 x_i、y_i 和一个低位来的进位位 C_{i+1}；输出有两个：本位和 z_i 和向高位的进位位 C_i。输入和输出之间的逻辑关系可以通过表 2-6 表征。

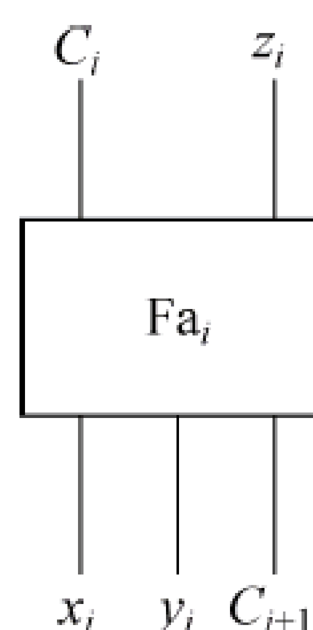


图 2-8 全加器

表 2-6 全加器真值表

输入			输出	
x_i	y_i	C_{i+1}	z_i	C_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

经逻辑设计化简，可得以下输入输出关系式

$$\begin{aligned} z_i &= x_i \oplus y_i \oplus C_{i+1} \\ C_i &= x_i y_i + (x_i + y_i) C_{i+1} \end{aligned} \tag{2.16}$$

构成一个多位的加法器可以将上述一位的全加器电路串联起来，将最低位的进位输入端置 0，其余位的进位输入端连接到低一位全加器的进位输出端。低位的进位输出像波的

传播一样传递到高位，这样的加法器电路称为行波进位加法器，或串行进位加法器。能同时实现补码加、减法的加法器，如图 2-9 所示。

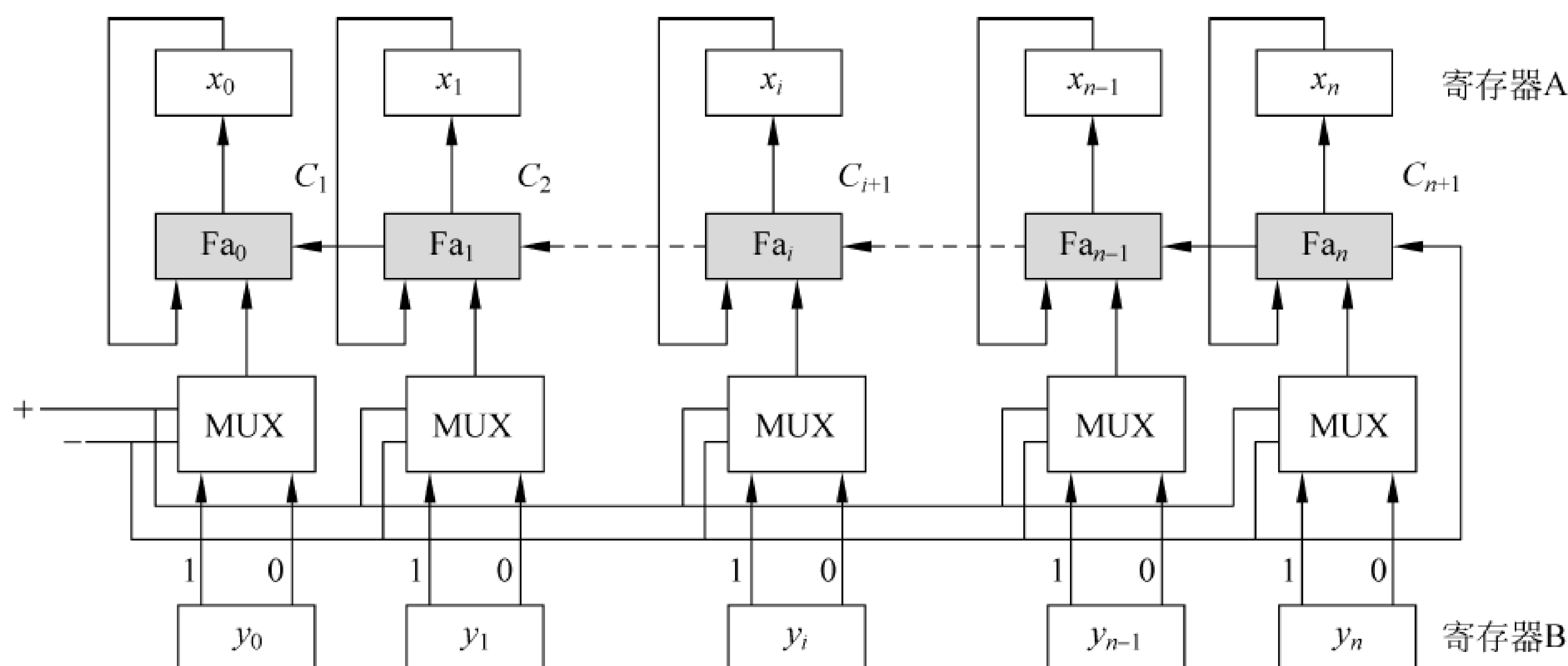


图 2-9　补码运算加法器

图 2-9 中，有两个 $n+1$ 位的寄存器 A 和 B，一个 $n+1$ 位的二选一多路开关 MUX 和 $n+1$ 个全加器。参加运算的两个相加数事先分别存放在寄存器 A 和 B 中，运算结果存放在 A 中。

当进行补码加时，“＋”、“－”控制端为 10，控制选择寄存器 B 的“1”端输出到全加器的一个输入端，实现 $x_0x_1\cdots x_i\cdots x_{n-1}x_n$ 加 $y_0y_1\cdots y_i\cdots y_{n-1}y_n$（此时 $C_{n+1}=0$），即 $[x]_补+[y]_补$。而当进行补码减时，“＋”、“－”控制端为 01，控制选择寄存器 B 的“0”端输出到全加器的一个输入端，实现 $x_0x_1\cdots x_i\cdots x_{n-1}x_n$ 加 $y_0y_1\cdots y_i\cdots y_{n-1}y_n$ 的反码并末位加 1（此时 $C_{n+1}=1$），即 $[x]_补+[-y]_补$。

行波进位加法器逻辑关系清晰，电路实现简单。但由于其进位位是由低到高一位一位串行生成的，因此，电路延迟时间长，运算速度慢。一种改进的方法是采用先行进位法，具体做法是：将 n 位相加的二进制位进行分组，每若干位分成一组，组内所有位的进位位同时生成，组间则仍然是串行进位。通过这种多级分组的方法，可以大大提高加法器的运算速度。

4. 十进制加法器的实现

二进制编码的十进制数可以直接运算。但要注意的是，其运算结果需要进行修正。具体修正规则是：如果两个一位 BCD 码相加之和小于或等于 $(1001)_2$，即十进制 9，则不需要修正；如相加之和大于 $(1001)_2$，则需要进行加 6 修正，并向高位进位。

【例 2.17】 ① 2＋6＝8　　　　② 4＋9＝13

①	②	
0010	0100	
＋0110	＋1001	
1000	1101	
不需要修正	＋0110	加 6 修正
	10011	

因此，十进制加法器可在二进制加法器的基础上加上适当的“校正”逻辑来实现，该校正逻辑可将二进制的“和”改变成所要求的十进制格式。图 2-10 是实现一位十进制 BCD 码加法的单元电路。

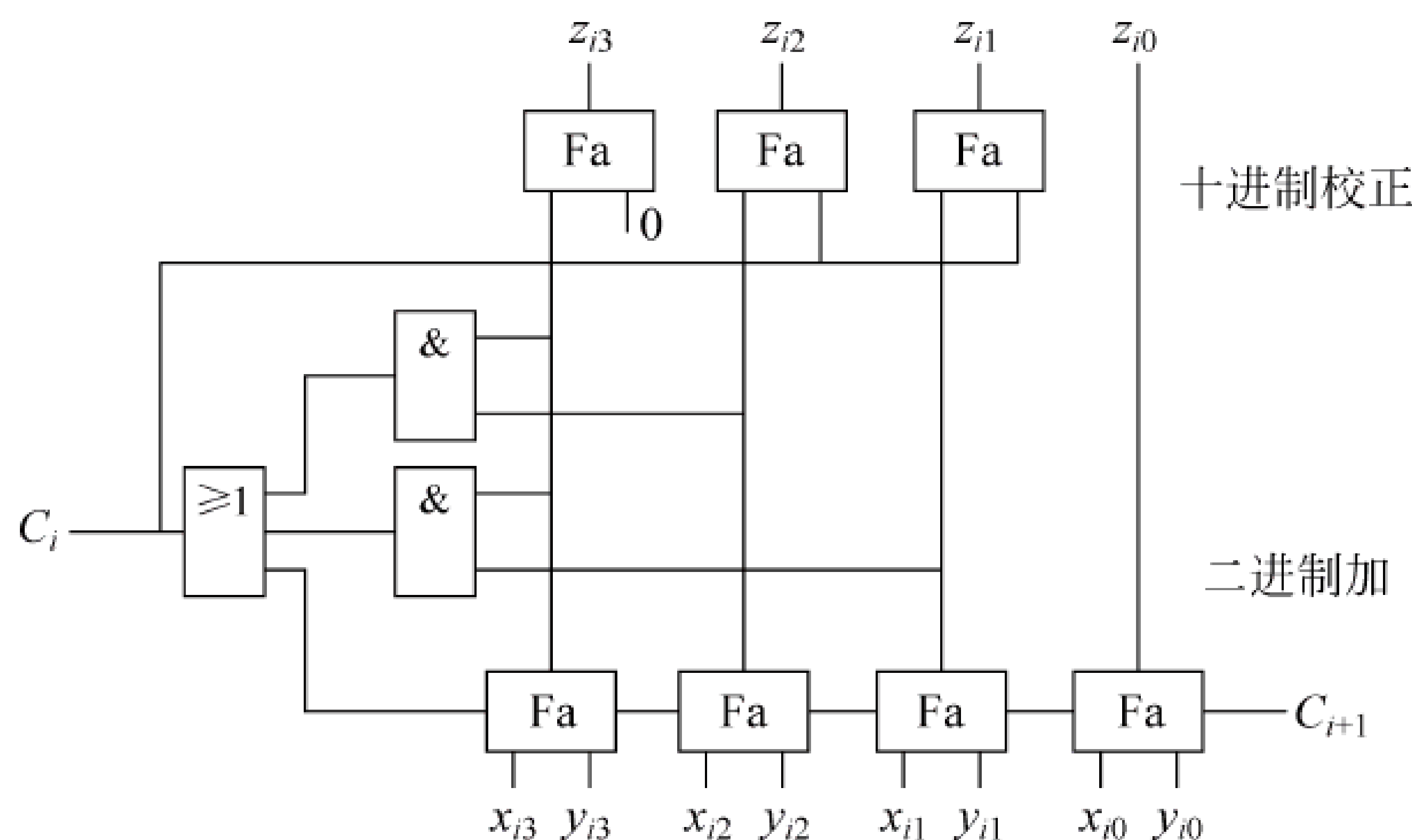


图 2-10　十进制 BCD 码加法单元电路

5．加减运算的溢出判别

下面给出两个补码运算的例子。

【例 2.18】　设 $x=0.1101, y=0.0110$,则有

$[x+y]_{补}=[x]_{补}+[y]_{补}=[0.1101]_{补}+[0.0110]_{补}=0.1101+0.0110=1.0011$

两个正数相加结果怎么变成了负数?

【例 2.19】　设 $x=-0.1011, y=-0.1001$,则有

$$[x+y]_{补}=[x]_{补}+[y]_{补}=[-0.1011]_{补}+[-0.1001]_{补}$$
$$=1.0101+1.0111=0.1100$$

两个负数相加结果怎么变成了正数?

以上两个运算的结果肯定是错误的,原因何在呢?

其实,在计算机中,任何种类的数据表示由于受到计算机字长的限制,其表示的数据都是有一定范围的。例如,8 位二进制补码表示的纯整数的范围是－128～＋127(即 10000000～01111111);16 位二进制补码表示的纯整数的范围是－4096～＋4095(即 1000000000000000～0111111111111111)。如果运算结果超出了表示范围,则称为产生了溢出。其中,若运算结果比最大的正数还大,则为正溢出;若运算结果比最小的负数还小,则为负溢出。

显然,当发生溢出时,其运算结果已不正确了,再继续后续的运算已毫无意义了。那么,机器是如何判断是否产生了溢出呢?下面介绍三种溢出的判别方法。

(1)符号位判别法。

从前面的两个例子可以注意到:两个正数相加的结果为负数,说明产生了溢出;两个负数相加结果为正数,也说明产生了溢出。将这两点归纳起来就是:当符号相同的两个数相加时,如果结果的符号与相加数的符号不同,则为溢出,而一正一负的两个数相加是不会产生溢出的。

对此结论,可以用逻辑关系式表达。设 x_0、y_0 分别为两个相加数的符号位,z_0 为运算结果的符号位,则溢出条件为

$$V=\overline{x_0}\,\overline{y_0}z_0+x_0y_0\overline{z_0} \tag{2.17}$$

若 $V=1$,则说明产生了溢出;若 $V=0$,则无溢出。

如图 2-11 所示是采用符号位判别法进行溢出判别的逻辑电路图。

（2）进位位判别法。

再来考察一下上述两个例子。对例 2.18、例 2.19 分别列竖式计算如下：

$$\begin{array}{r} 0.1101 \\ +0.0110 \\ \hline \underset{\text{有进位}}{1}.\underset{\text{无进位}}{0}011 \end{array} \qquad \begin{array}{r} 1.0101 \\ +1.0111 \\ \hline \underset{\text{无进位}}{0}.\underset{\text{有进位}}{1}100 \end{array}$$

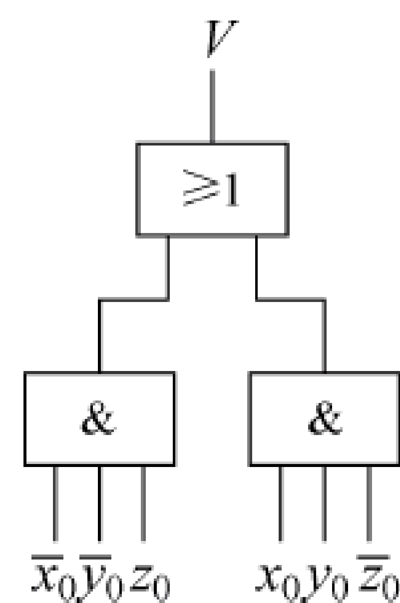


图 2-11 符号位判别法

设两相加数的结果的符号位和数值部分最高位的进位位分别为 C_0、C_1。当两个正数相加且发生溢出时，$C_0=1$，$C_1=0$，即符号位产生了进位，而数值部分最高位未产生进位；当两个负数相加且发生溢出时，$C_0=0$，$C_1=1$，即符号位未产生进位，而数值部分最高位产生了进位。将这两点归纳起来就是：当两个补码相加时，如果符号位和数值部分最高位的进位位 C_0C_1 不同，则为溢出，相同则未溢出。

对此结论，可以用逻辑关系式表达。设 C_0、C_1 分别为结果的符号位和数值部分最高位的进位位，则溢出条件为

$$V = C_0 \oplus C_1 \tag{2.18}$$

若 $V=1$，则说明产生了溢出；若 $V=0$，则无溢出。

如图 2-12 所示是采用进位位判别法进行溢出判别的逻辑电路图。

（3）双符号位法。

两个相加数均使用两位符号位，00 表示正数，11 表示负数。对例 2.18、例 2.19 分别使用两位符号位列竖式计算如下：

$$\begin{array}{r} 00.1101 \\ +00.0110 \\ \hline 01.0011 \end{array} \qquad \begin{array}{r} 11.0101 \\ +11.0111 \\ \hline 10.1100 \end{array}$$

可以得出以下结论：当两个正数相加时，若结果的两个符号位相同，则无溢出；若不同，则有溢出，且为 01 时是正溢出，为 10 时是负溢出。

设运算结果的两个符号位为 z_0z_0'，则溢出条件为

$$V = z_0 \oplus z_0' \tag{2.19}$$

若 $V=1$，则说明产生了溢出；若 $V=0$，则无溢出。

图 2-13 是采用双符号位判别法进行溢出判别的逻辑电路图。

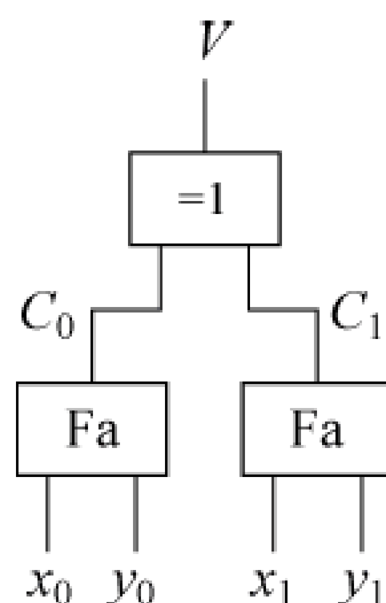


图 2-12 进位位判别法

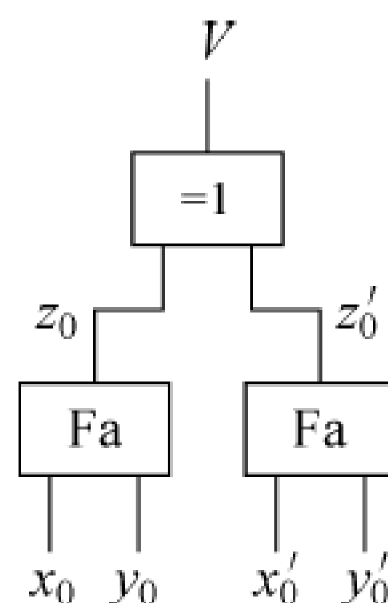


图 2-13 双符号位判别法

使用双符号位的补码又称补码变形码，其纯小数定义为

$$[x]_{补} = 4 + x \quad (\text{mod } 4) \tag{2.20}$$

其纯整数定义为

$$[x]_{补} = 2^{n+2} + x \quad (\text{mod } 2^{n+2}) \tag{2.21}$$

2.4.2 定点数的乘法运算

1. 原码一位乘法

原码乘法是将符号位与数值部分分开进行运算，运算结果的数值部分是两相乘数的数值部分之积，而符号位则是两相乘数的符号位的异或。

设$[x]_{原} = x_0x_1x_2\cdots x_n$，$[y]_{原} = y_0y_1y_2\cdots y_n$，则

$$[xy]_{原} = (x_0 \oplus y_0) \mid (x_1x_2\cdots x_n)(y_1y_2\cdots y_n) \tag{2.22}$$

为了在计算机中实现原码一位乘法，先考察一下手工完成二进制乘的方法和步骤，见例 2.20。

【例 2.20】 $x=0.1001$，$y=0.1101$，计算 xy。

解：

```
       1001
     ×1101
    ---------
       1001      位积 1
      0000       位积 2
     1001        位积 3
    1001         位积 4
    ---------
    01110101
```

即 $xy=0.01110101$。

手工计算时，是将乘数 y 从低到高逐位乘以被乘数 x，每次得到的位积都相对上一个位积左移一位，最后将所有位积一次性相加，即得乘积。

考虑到机器硬件的特点，在计算机中实现乘法运算时，必须对手工过程进行以下调整：

(1) 硬件实现时需要使用三个寄存器分别存放乘数、被乘数和部分积。

(2) 机器中的运算器一般一次只能完成两个数的相加，手工一次性相加可以改为逐次加。即当得到两个位积时，进行一次相加，得到一个部分积；以后每得到一个位积，都将其与上次的部分积相加，得到新的部分积；最后一次的部分积即为乘积。

(3) 手工计算时，每次得到的位积都相对上一次位积左移一位，由于最后的乘积是乘数或被乘数的两倍，因此，加法器的位数也必须加倍。通过观察手工计算可以看出，每次部分积的最低位并不参加运算。因此，在计算机实现时，可以每次将部分积右移一位，部分积的最低位直送即可。这样的话，加法器的位数与乘数或被乘数的位数相同。

(4) 部分积右移时，将乘数寄存器同时右移一位，这样一方面可以每次根据乘数寄存器的值决定本次位积的值(若最低位为 1，则本次位积为被乘数；若最低位为 0，则本次位积为 0)；另一方面，乘数寄存器的最高位每次可以接受部分积右移出来的一位。

(5) 运算完成后，部分积寄存器和乘数寄存器中分别存放的是最后乘积的高半部和低半部。

如例 2.21 是经过以上调整后计算机实现原码一位乘的过程。

【例 2.21】 $x=0.1001$，$y=0.1101$，计算 xy。

解：部分积初值为 0，且使用双符号位，运算过程如下：

	部分积	乘数
	00　0000	1　1　0　1
$+x$	00　1001	
	00　1001	
右移 1 位	00　0100	1　1　1　0
$+0$	00　0000	
	00　0100	
右移 1 位	00　0010	0　1　1　1
$+x$	00　1001	
	00　1011	
右移 1 位	00　0101	1　0　1　1
$+x$	00　1001	
	00　1110	
右移 1 位	00　0111	0　1　0　1
	乘积高位	乘积低位

即 $xy=0.01110101$。

计算机中的原码一位乘可以通过循环迭代的方法实现，其中每次得到的部分积 P_i 为

$$P_0=0$$
$$P_1=(P_0+xy_n)2^{-1}$$
$$P_2=(P_1+xy_{n-1})2^{-1}$$
$$\vdots$$
$$P_{i+1}=(P_i+xy_{n-i})2^{-1}$$
$$\vdots$$
$$P_n=(P_{n-1}+xy_1)2^{-1}$$

P_n 为乘积。

实现原码一位乘法的逻辑电路原理框图如图 2-14 所示。

图 2-14 中，R0、R1、R2 三个寄存器分别存放被乘数、部分积和乘数，其中 R1 的初值为 0。当乘法开始时，根据 R2 的最低位 y_n 的值决定本次是将 R0 的内容还是将 0 与 R1（上次部分积）相加，结果送回 R1，然后将 R1、R2 联合右移一位，得到新的部分积。如此重复 n 步，最后的乘积存在 R1、R2 中。

2. 补码一位乘法

数据在计算机中是以补码的形式存储的，在用原码实现乘法时，需要先将补码转换成原码，相乘之后再将原码的结果转换回补码。为了避免这种转换，很多计算机采用补码直

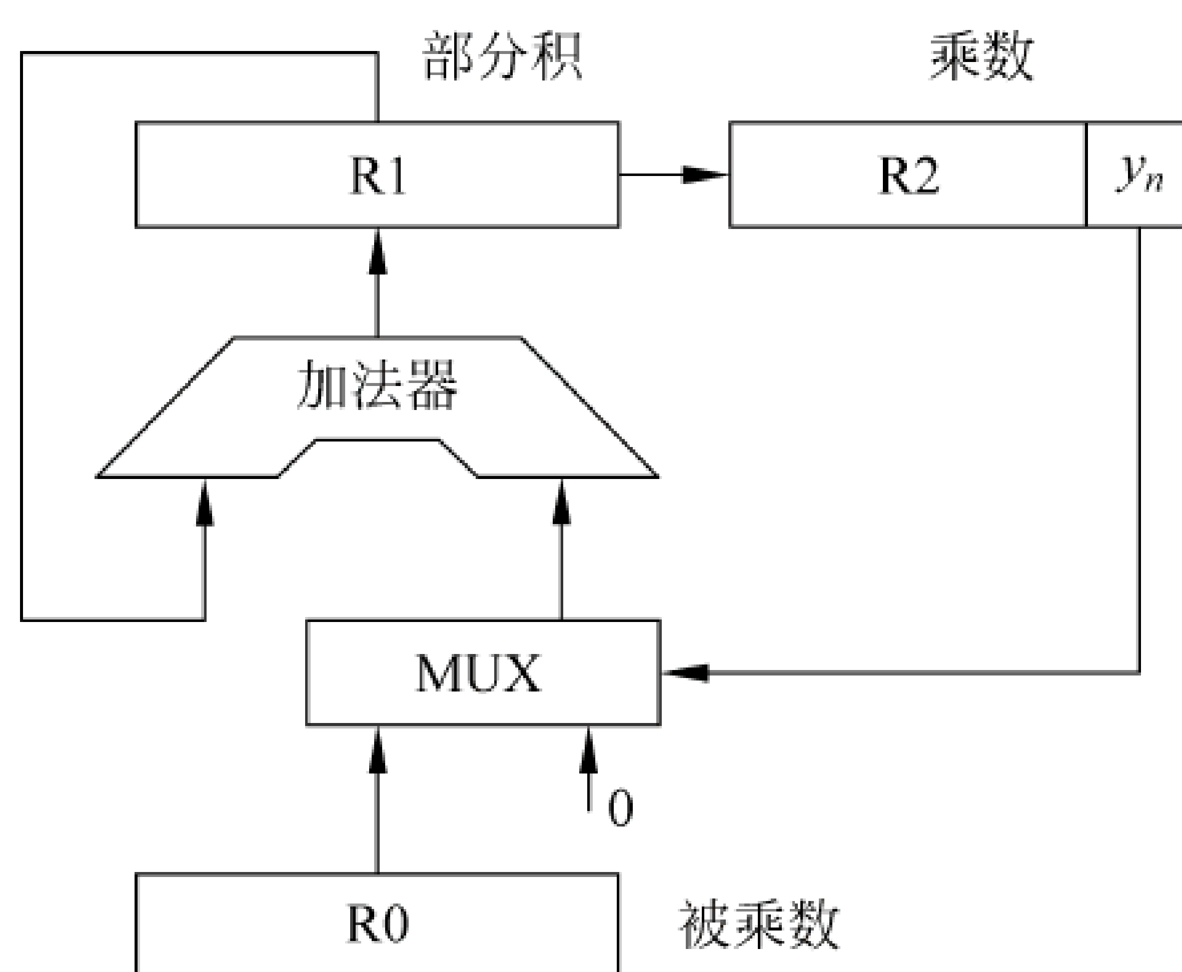


图 2-14 原码一位乘法逻辑电路原理框图

接乘。

与原码一位乘不同的是，补码一位乘的两个乘数和被乘数的符号位是直接参加运算的。下面介绍一种布斯乘法，该算法是由布斯(Booth)最早提出的，故以其名字命名。

设$[x]_补=x_0.x_1x_2\cdots x_n$，$[y]_补=y_0.y_1y_2\cdots y_n$，则有

$$
\begin{aligned}
[xy]_补 &= [x]_补(-y_0+0.y_1y_2\cdots y_n)\\
&= [x]_补(-y_0+2^{-1}y_1+2^{-2}y_2+\cdots+2^{-n}y_n)\\
&= [x]_补[-y_0+(y_1-2^{-1}y_1)+(2^{-1}y_2-2^{-2}y_2)+\cdots+(2^{-n+1}y_n-2^{-n}y_n)]\\
&= [x]_补[(y_1-y_0)+2^{-1}(y_2-y_1)+\cdots+2^{-n+1}(y_n-y_{n-1})+2^{-n}(0-y_n)] \quad (2.23)
\end{aligned}
$$

根据式(2.23)可以通过迭代公式计算补码一位乘，其中每次得到的部分积P_i为

$$
\begin{aligned}
[P_0]_补 &= 0\\
[P_1]_补 &= \{[P_0]_补+(y_{n+1}-y_n)[x]_补\}2^{-1} \qquad y_{n+1}=0\\
[P_2]_补 &= \{[P_1]_补+(y_n-y_{n-1})[x]_补\}2^{-1}\\
&\vdots\\
[P_i]_补 &= \{[P_{i-1}]_补+(y_{n-i+2}-y_{n-i+1})[x]_补\}2^{-1}\\
&\vdots\\
[P_n]_补 &= \{[P_{n-1}]_补+(y_2-y_1)[x]_补\}2^{-1}\\
[xy]_补 &= [P_{n+1}]_补=[P_n]_补+(y_1-y_0)[x]_补
\end{aligned}
$$

由此得到布斯一位乘法的运算步骤如下：

(1) 初始部分积为0。

(2) 根据乘数寄存器的最低两位决定：若为00或11，则将上次部分积直接右移一位；若为01，则将上次部分积加$[x]_补$，然后右移一位；若为10，则将上次部分积加$[-x]_补$，然后右移一位。得到新部分积。

(3) 如此重复$n+1$步，最后一步不移位。

【例 2.22】 $x=0.1001$，$y=0.1101$，计算$[xy]_补$。

解：计算过程如下。

	部分积	乘数	
	00 0000	0. 1 1 0 1 0	末尾补 0
$+[-x]_补$	11 0111		
	11 0111		
右移 1 位	11 1011	1 0 1 1 0 1	
$+[x]_补$	00 1001		
	00 0100		
右移 1 位	00 0010	0 1 0 1 1 0	
$+[-x]_补$	11 0111		
	11 1001		
右移 1 位	11 1100	1 0 1 0 1 1	
右移 1 位	11 1110	0 1 0 1 0 1	
$+[x]_补$	00 1001		
	00 0111		
$[x \cdot y]_补$ =	00 0111	0 1 0 1	最后一步不移位

为了提高乘法的运算速度，计算机设计者提出了各种并行算法，如两位乘法、多位乘法、阵列乘法等，读者有兴趣可参阅相关资料。

2.4.3 定点数的除法运算

1. 原码一位除法

原码除法是将符号位与数值部分分开进行运算，商的数值部分是两相除数的数值部分相除之后的结果，而商的符号位则是两相除数的符号位的异或。

设$[x]_原=x_0x_1x_2\cdots x_n$，$[y]_原=y_0y_1y_2\cdots y_n$，则

$$[x/y]_原=(x_0 \oplus y_0) \mid (x_1x_2\cdots x_n)/(y_1y_2\cdots y_n) \tag{2.24}$$

为了在计算机中实现原码一位除法，先考察一下手工完成二进制除的方法和步骤，如例 2.23 所示。

【例 2.23】 $x=0.1011$，$y=0.1101$，计算 x/y。

解：

```
          1101                         0.1101
1101 ) 10110                 0.1101 ) 0.10110
        1101                           0.01101
       ------                          ----------
        10010                          0.010010
         1101          或者            0.001101
       ------                          ----------
         10100                         0.00010100
          1101                         0.00001101
       ------                          ----------
          0111                         0.00000111
```

即商为 0.1101，余数为 0.00000111。

手工进行二进制除法的方法是：判断余数（第一次为被除数）与除数的大小，若余数小，则商为 0，并在余数末尾补 0，再用余数和右移一位的除数比较，若余数大，则商 1，并做减

法，得新余数。重复上述步骤，直到除尽或已得到的商的位数满足精度要求为止。

在计算机中实现原码一位除需要对手工过程做以下的改进：

(1) 硬件实现时需要使用三个寄存器分别存放被除数(余数)、除数和商。

(2) 为使加法器的位数不增加，可以将手工中的右移除数改为左移余数，左移出去的余数的高位都是无用的 0，对运算不会产生任何影响。

(3) 手工中的商 0 或 1 是通过计算者用观察比较的办法确定的，而在计算机中，只能用做减法后判断结果的符号位来确定。当差为负时，则商为 0，同时还应把除数加回到差上去，恢复余数为原来的正值之后再将其左移一位。若差为正，则商为 1，并将余数左移一位。这种方法称为原码一位除的恢复余数法。

恢复余数法的缺点是：每次当将余数减去除数得到的差为负数时，又要将除数加回到差上去，多增加了运算步骤。

对恢复余数法改进的一种方法是加减交替法，其运算规则为：每次将余数(第一次为被除数)减去除数，若余数为正，则商为 1，并将余数左移一位，减除数，得新余数；若余数为负，则商为 0，并将余数左移一位，加除数，得新余数。

例 2.24 是原码一位除的加减交替法的运算过程。

【例 2.24】 $x=0.1011$，$y=0.1101$，计算 x/y。

解：商初值为 0，被除数(余数)使用双符号位。

$$[y]_{补}=00.1101,\quad [-y]_{补}=11.0011$$

	被除数(余数)	商	
	00 1011	0. 0 0 0 0	
$+[-y]_{补}$	11 0011		
	11 1110	0 0 0 0 0	差为负，商 0
左移 1 位	11 1100	0 0 0 0 0	
$+[y]_{补}$	00 1101		
	00 1001	0 0 0 0 1	差为正，商 1
左移 1 位	01 0010	0 0 0 1 0	
$+[-y]_{补}$	11 0011		
	00 0101	0 0 0 1 1	差为正，商 1
左移 1 位	00 1010	0 0 1 1 0	
$+[-y]_{补}$	11 0011		
	11 1101	0 0 1 1 0	差为负，商 0
左移 1 位	11 1010	0 1 1 0 0	
$+[y]_{补}$	00 1101		
	00 0111		
	00 0111	0 1 1 0 1	差为正，商 1

x/y 的商为 0.1101，余数为 0.0111。

实现原码一位除的逻辑电路原理框图如图 2-15 所示。

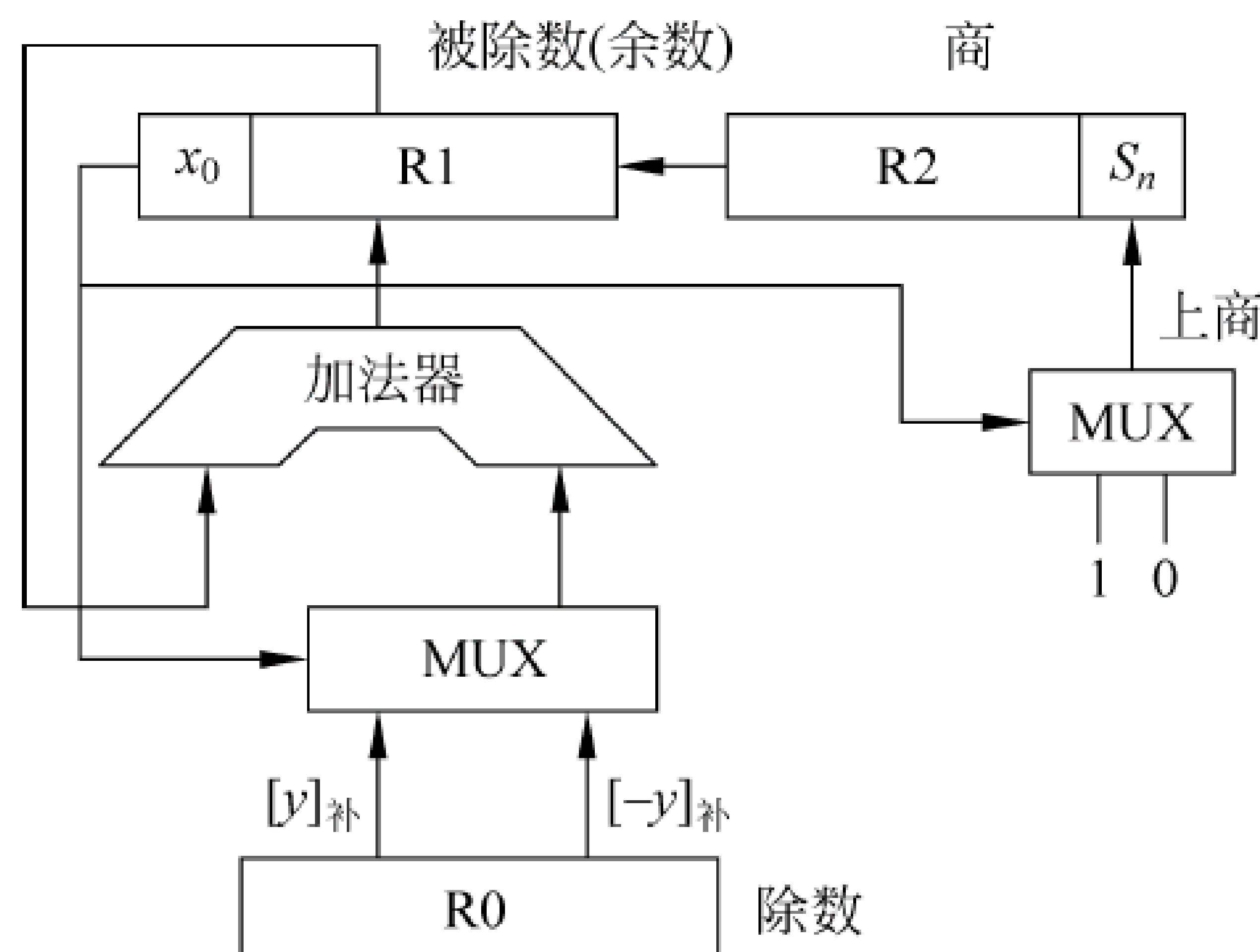


图 2-15 原码一位除的逻辑电路原理框图

2. 补码一位除法

进行补码一位除时，被除数和除数的符号位一起参加运算。

相对原码除来讲，补码除的运算规则更复杂。当除数和被除数用补码表示时，判别是否够减，要比较它们的绝对值的大小。因此，若两数同符号，要用减法，若异号，则要用加法。对于判断是否够减，及确定本次商 1 或商 0 的规则，还与结果的符号有关。当商为正时，商的每一位上的值与原码表示一致；而当商为负时，商的各位应是补码形式的值，一般先按各位的反码值上商，除完后，再用在最低位上加 1 的办法求出正确的补码值。

在被除数的绝对值小于除数的绝对值(即商未溢出)的情况下，补码一位除法的运算规则如下：

(1) 如果被除数与除数同号，用被除数减去除数；若两数异号，用被除数加上除数。如果所得余数与除数同号，则商 1，否则商 0，此即商的符号位。

(2) 求商的数值部分。如果上次商 1，将余数左移一位后减去除数；如果上次商 0，将余数左移一位后加上余数。然后判断本次操作后的余数，如果余数与除数同号，则商 1，否则商 0。

(3) 如此重复执行 $n-1$ 次(设数值部分有 n 位)。

补码一位除的例子就不再举了。

以上介绍的定点乘和定点除法都属于串行运算方法，即通过“加(减)—移位”操作来实现乘除运算。这些方法的特点是算法简单，硬件实现容易。在本书中介绍这些方法的目的是使读者能建立计算机运算的设计思想，初步了解机器硬件是如何实现一定的算法的。随着人们对机器运算速度的不断追求，计算机设计者提出了各种并行算法，以不断提高机器的乘除运算速度，从而提高 CPU 的整体性能。读者有兴趣可参阅相关资料。

2.5 浮点数的运算

浮点数由于是由阶码和尾数两个部分组成的，因此，浮点数的加、减和乘、除运算具有不同于定点数的运算方法，在溢出判别上也有所不同。

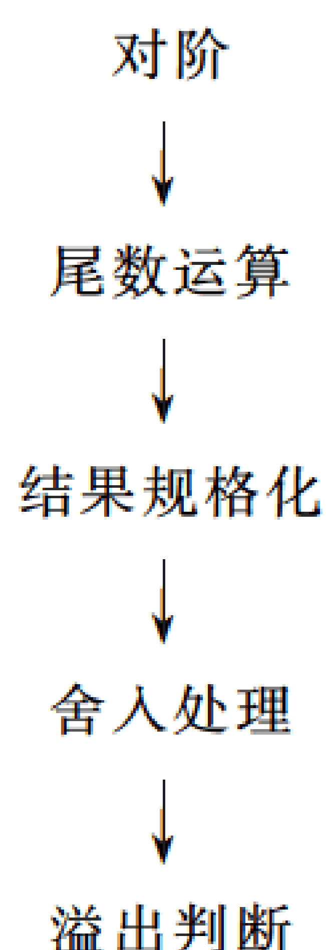
2.5.1 浮点数的加减运算

二进制浮点数的表示形式为

$$N=2^E M$$

其中,M 为浮点数的尾数,一般为纯小数,在计算机中通常采用补码表示;E 为阶码,为纯整数,用补码或移码表示。浮点数的运算实际上最终转化为定点数的运算来完成。但由于浮点数表示的特殊性,其运算又比单纯的定点数运算更复杂。

浮点数的加减运算一般由以下 5 个步骤完成:



设两浮点数 X、Y 进行加减运算,其中 $X=2^{E_x}M_x$,$Y=2^{E_y}M_y$。

1. 对阶

所谓对阶是指将两个进行运算的浮点数的阶码对齐的操作。对阶的目的是为使两个浮点数的尾数能够进行加减运算。因为,当进行 $2^{E_x}M_x$ 与 $2^{E_y}M_y$ 加减运算时,只有使两浮点数的指数值部分相同,才能将相同的指数值作为公因数提出来,然后进行尾数的加减运算。

对阶的具体方法是:首先求出两浮点数阶码的差,即 $\Delta E=E_x-E_y$,将小阶码加上 ΔE,使之与大阶码相等,同时将小阶码对应的浮点数的尾数右移相应位数,以保证该浮点数的值不变。几点注意:

(1) 对阶的原则是小阶对大阶,之所以这样做是因为若大阶对小阶,则尾数的数值部分的高位需移出,而小阶对大阶移出的是尾数的数值部分的低位,这样损失的精度更小。

(2) 若 $\Delta E=0$,说明两浮点数的阶码已经相同,无须再做对阶操作了。

(3) 采用补码表示的尾数右移时,符号位保持不变。

(4) 由于尾数右移时是将最低位移出,会损失一定的精度,为减少误差,可先保留若干移出的位,供以后舍入处理用。

2. 尾数运算

尾数运算就是完成对阶后的尾数相加减。这里采用的就是前面讲过的纯小数的定点数加减运算。

3. 结果规格化

在机器中，为保证浮点数表示的唯一性，浮点数在机器中都是以规格化形式存储的。对于 IEEE 754 标准的浮点数来说，就是尾数必须是 1.*M* 的形式。由于在进行上述两个定点小数的尾数相加减运算后，尾数有可能是非规格化形式，为此必须进行规格化操作。

规格化操作包括左规和右规两种情况。

左规操作：将尾数左移，同时阶码减值，直至尾数成为 1.*M* 的形式。例如，浮点数 0.0011×2^5 是非规格化的形式，需进行左规操作，将其尾数左移 3 位，同时阶码减 3，就变成 1.1100×2^2 规格化形式了。

右规操作：将尾数右移 1 位，同时阶码增 1，便成为规格化的形式了。要注意的是，右规操作只需将尾数右移一位即可，这种情况出现在尾数的最高位(小数点前一位)运算时出现了进位，使尾数成为 10.xxxx 或 11.xxxx 的形式。例如，10.0011×2^5 右规一位后便成为 $1.0001\underline{1}\times 2^6$ 的规格化形式了。

4. 舍入处理

浮点运算在对阶或右规时，尾数需要右移，被右移出去的位会被丢掉，从而造成运算结果精度的损失。为了减少这种精度损失，可以将一定位数的移出位先保留起来，称为保护位，在规格化后用于舍入处理。

IEEE 754 标准列出了 4 种可选的舍入处理方法：

(1) 就近舍入(round to nearest)。

这是标准列出的默认舍入方式，其含义相当于日常所说的“四舍五入”。例如，对于 32 位单精度浮点数来说，若超出可保存的 23 位的多余位大于等于 100…01，则多余位的值超过了最低可表示位值的一半，这种情况下，舍入的方法是在尾数的最低有效位上加 1；若多余位小于等于 011…11，则直接舍去；若多余位为 100…00，此时再判断尾数的最低有效位的值，若为 0 则直接舍去，若为 1 则再加 1。

(2) 朝 $+\infty$ 舍入(round toward $+\infty$)。

对正数来说，只要多余位不为全 0，则向尾数最低有效位进 1；对负数来说，则是简单地舍去。

(3) 朝 $-\infty$ 舍入(round toward $-\infty$)。

与朝 $+\infty$ 舍入方法正好相反，对正数来说，只是简单地舍去；对负数来说，只要多余位不为全 0，则向尾数最低有效位进 1。

(4) 朝 0 舍入(round toward 0)。

即简单地截断舍去，而不管多余位是什么值。这种方法实现简单，但容易形成累积误差，且舍入处理后的值总是向下偏差。

5. 溢出判断

与定点数运算不同的是，浮点数的溢出是以其运算结果的阶码的值是否产生溢出来判断的。若阶码的值超过了阶码所能表示的最大正数，则为上溢，进一步，若此时浮点数为正数，则为正上溢，记为 $+\infty$，若浮点数为负数，则为负上溢，记为 $-\infty$；若阶码的值超过了阶

码所能表示的最小负数,则为下溢,进一步,若此时浮点数为正数,则为正下溢,若浮点数为负数,则为负下溢。正下溢和负下溢都作为 0 处理。

要注意的是,浮点数的表示范围和补码表示的定点数的表示范围是有所不同的,定点数的表示范围是连续的,而浮点数的表示范围可能是不连续的,如图 2-16 所示。

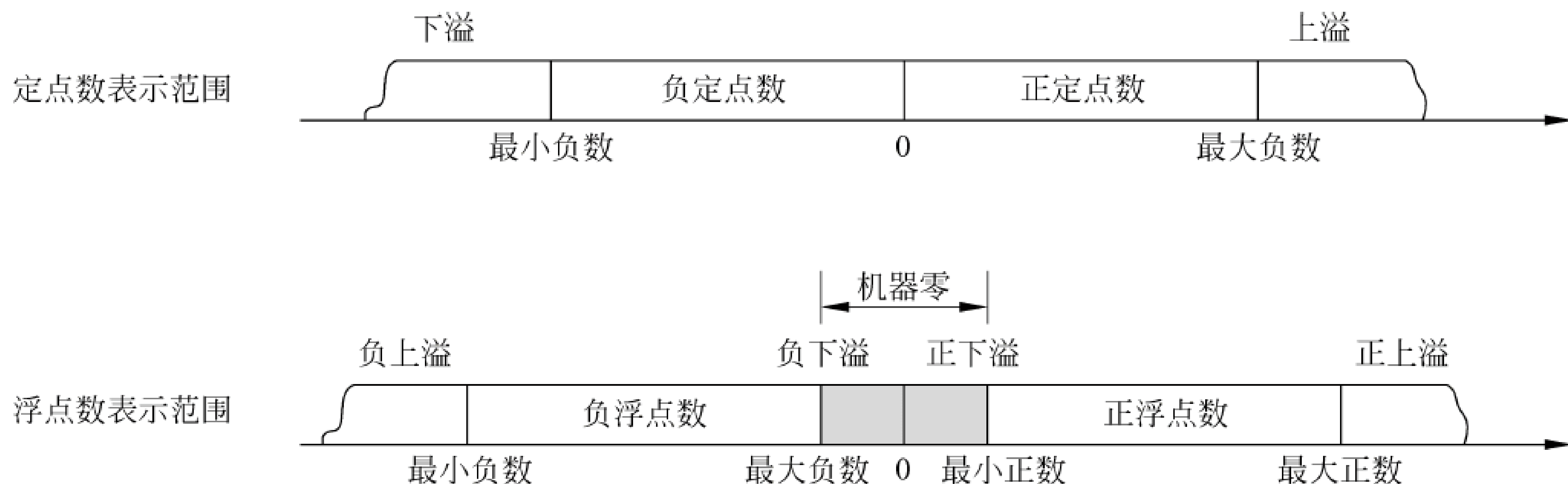


图 2-16 定点数和浮点数表示范围

【例 2.25】 设两浮点数的 IEEE 754 标准存储格式分别为

x=0 10000010 01101100000000000000000,y=0 10000100 01011101100000000000000,求 $x+y$,并给出结果的 IEEE 754 标准存储格式。

解:对于浮点数 x:

符号位 $S=0$

指数 $e=E-127=10000010-01111111=00000011=(3)_{10}$

尾数 $m=1.M=1.01101100000000000000000=1.011011$

于是有

$$x=(-1)^s\times m\times 2^e=+1.01101100000000000000000\times 2^3$$

对于浮点数 y:

符号位 $S=0$

指数 $e=E-127=10000100-01111111=00000011=(5)_{10}$

尾数 $m=1.M=1.01011101100000000000000=1.010111011$

于是有

$$y=(-1)^s\times m\times 2^e=+1.01011101100000000000000\times 2^5$$

(1) 对阶。

$E=E_x-E_y=3-5=-2$

$x=1.01101100000000000000000\times 2^3=0.01011011000000000000000\ \underline{00}\times 2^5$

(2) 尾数相加。

$$\begin{aligned}x+y&=0.01011011000000000000000\ \underline{00}\times 2^5+1.01011101100000000000000\times 2^5\\&=1.10111000100000000000000\times 2^5\end{aligned}$$

结果的 IEEE 754 标准存储格式为 0 10000100 10111000100000000000000。

实现浮点运算的加法器逻辑电路原理框图如图 2-17 所示。

图 2-17 中,三个寄存器 R0、R1 和 R2 分别存放两个参加运算的浮点数和结果。第一步对阶,首先由 ΔE 加法器求出两个浮点数阶码的差值,然后由控制电路控制选择小阶码浮点

数的尾数进入右移寄存器进行对阶时的右移，右移结果送入尾数加法器的一个输入端，大阶码浮点数的尾数则直接送入加法器的另一个输入端。第二步尾数相加减。第三步规格化，由尾数加法器产生的结果经规格化部件，一方面送移位寄存器进行尾数移位，另一方面控制选择大阶码进行阶码的增或减操作。第四步由舍入部件对规格化后的尾数进行舍入处理，并将结果送结果寄存器的尾数字段。第五步溢出处理，由溢出判别部件对规格化后的阶码进行溢出判别，若未溢出，则将结果送结果寄存器的阶码部分。

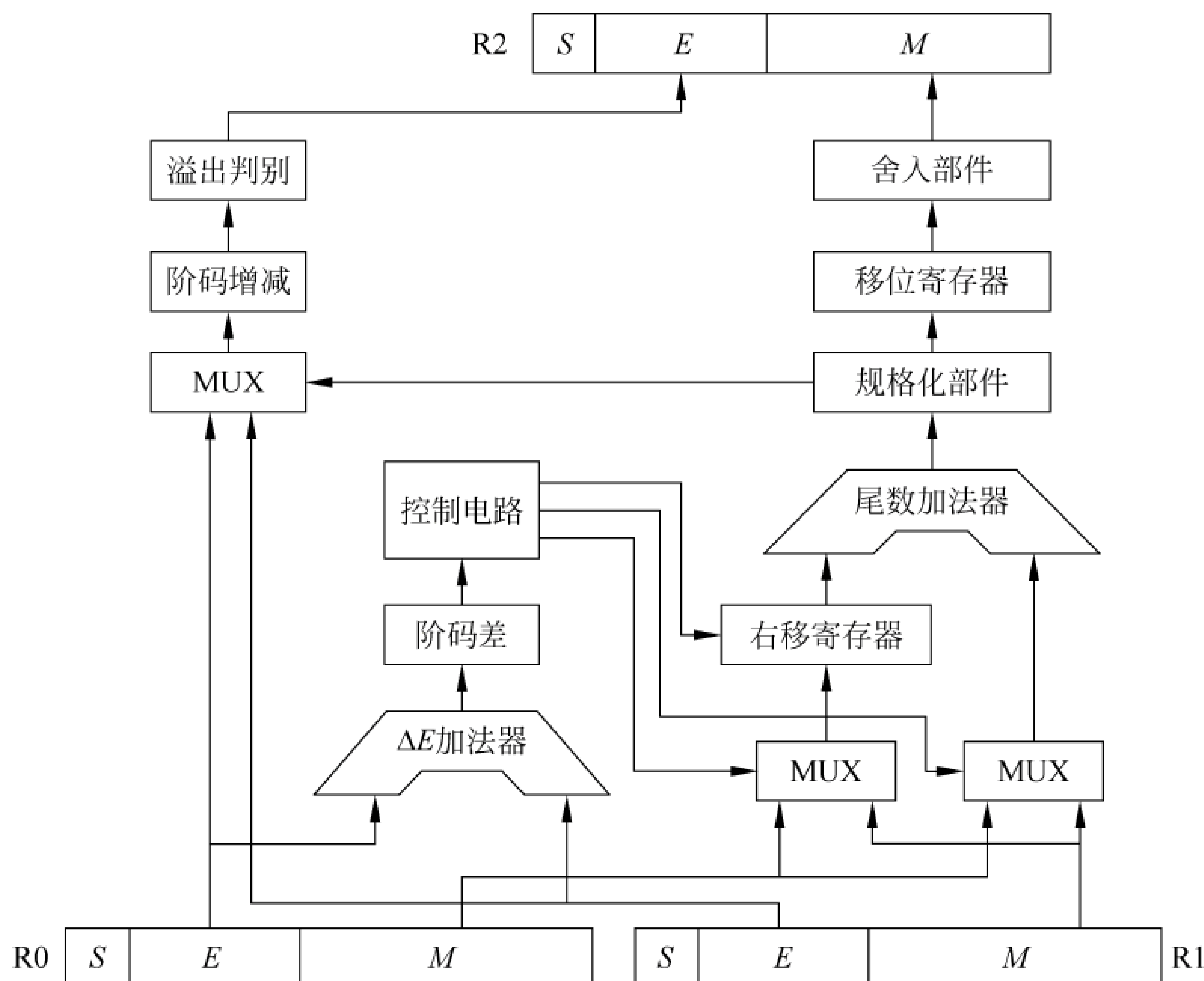


图 2-17　浮点运算加法器逻辑电路原理框图

2.5.2　浮点数的乘除运算

设两个浮点数 $X=2^{E_x}M_x$，$Y=2^{E_y}M_y$，则

$$
\begin{aligned}
XY &= (2^{E_x}M_x)(2^{E_y}M_y) \\
&= 2^{E_x+E_y}(M_x \times M_y)
\end{aligned}
$$

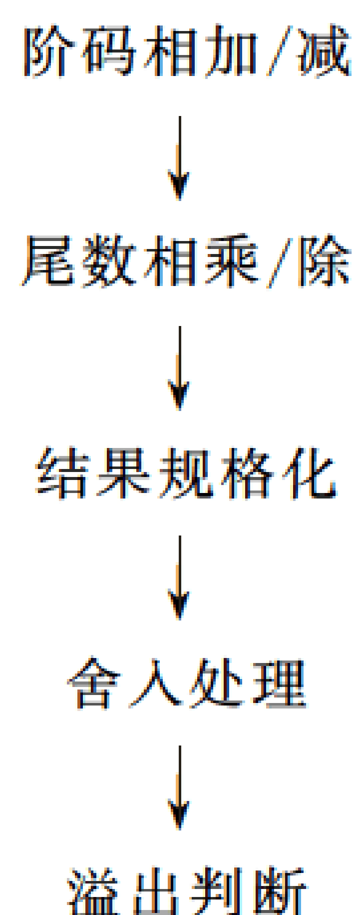
由此可见，浮点数乘法的基本规则是：两浮点数相乘，乘积的阶码是相乘两数阶码之和，乘积的尾数是相乘两数尾数之积。简单来说就是：阶码相加，尾数相乘。

同样

$$
\begin{aligned}
X/Y &= (2^{E_x}M_x)/(2^{E_y}M_y) \\
&= 2^{E_x-E_y}(M_x \times M_y) \\
&= 2^{E_x+E_y}(M_x \times M_y)
\end{aligned}
$$

由此可得浮点数除法的基本规则是：两浮点数相除，商的阶码是相除两数阶码之差，商的尾数是相除两数尾数之商。简单来说就是：阶码相减，尾数相除。

浮点数的乘除运算比加减运算少了对阶这一步，一般由以下 5 个步骤完成：



其中，阶码相加/减进行的是定点整数的加、减运算，尾数相乘/除进行的是定点小数的乘、除运算。结果规格化、舍入处理和溢出判断等与前面的浮点数的加减运算相同，此不再赘述。

2.6 数据校验码

数据在计算机系统中的生成、处理、存储和传输过程中都可能会发生错误，例如将数据存入存储器中产生的错误，数据在数据通道中传输时产生的错误等。这些错误都属于硬件错误，又称为物理差错。虽然可以通过提高硬件的可靠性等手段减少和避免这种错误，但不管怎样，都无法将发生物理差错的概率降为零，尤其是现代机器要求数据存储的密度越来越高，数据传输速度越来越快，这在一定程度上会加大发生数据差错的可能性。

从物理差错的种类上讲，主要分为随机错和突发错。随机错是指由于硬件设备或物理传输介质自身原因产生的差错，这种错误是随机发生的，往往是一次发生一位或若干位的错误。突发错是指由于外界的干扰产生的差错，这种错误是突发的，往往是一次发生大片数据的差错。

为了解决这种物理差错的问题，一种常用的方法是数据编码，即对要存储或传输的数据进行编码，使之具有检测或纠正差错的能力，这种编码称为数据校验码。数据校验码的基本思想是：在数据位的基础上，额外增加若干位的冗余位（又称校验位），使编码后的数据符合某种规律，符合这种规律的编码属于合法码，不符合这种规律的编码属于非法码，通过检测一个数据编码的合法性，就可以判断差错的发生，进一步进行差错的定位，从而纠正错误。具有检测错误能力的编码称为检错码，具有纠正错误能力的编码称为纠错码。常用的数据校验码包括奇偶校验码、海明校验码和循环冗余码等。

2.6.1 奇偶校验码

奇偶校验码是一种最简单的数据校验码，是奇校验码和偶校验码的统称，属于检错码。它的编码规则是：在数据位的基础上增加一位冗余位，使数据编码码的合法码中1的个数恒为奇数或偶数。若为奇数，则为奇校验码；若为偶数，则为偶校验码。例如：

数据位：1 0 0 1 1 1 0 1→1 0 0 1 1 1 0 1 1 偶校验码

数据位：1 0 0 1 1 1 0 1→1 0 0 1 1 1 0 1 0 奇校验码

其中1、0为冗余位。

从检纠错能力上讲，奇偶校验码可以检测发生奇数位错的情况，但不能纠错。由于同时发生多位错的概率远比发生一位错的低，所以一般认为奇偶校验码具有检测发生一位错的能力，但不能进行错误定位，即不能纠正错误。

下面以偶校验为例，对奇偶校验码的校验方法和实现原理进行进一步讨论。

设8位数据位 $D=D_1D_2D_3D_4D_5D_6D_7D_8$，则偶校验编码为 $D_1D_2D_3D_4D_5D_6D_7D_8P$，其中偶校验位

$$P=D_1\oplus D_2\oplus D_3\oplus D_4\oplus D_5\oplus D_6\oplus D_7\oplus D_8$$

偶校验检测位为

$$P^*=D_1\oplus D_2\oplus D_3\oplus D_4\oplus D_5\oplus D_6\oplus D_7\oplus D_8\oplus P$$

若 $P^*=0$，则无错；若 $P^*=1$，则有错。

【例2.26】 设数据位 $D_1D_2D_3D_4D_5D_6D_7D_8=01001110$，求其偶校验编码。

解：$P=D_1\oplus D_2\oplus D_3\oplus D_4\oplus D_5\oplus D_6\oplus D_7\oplus D_8=0\oplus 1\oplus 0\oplus 0\oplus 1\oplus 1\oplus 1\oplus 0=0$

因此，偶校验编码为01001110 0。

奇偶校验码常用于计算机中的内存校验。具体做法是：以字节为单位进行编码，即每8位二进制数据位附加1位校验位，共9位编码。每次将9位编码写入内存单元中，今后每读出的9位数据若为合法码，则说明未发生差错，若为非法码，则说明发生了差错。

可以通过如图2-18所示的电路分别实现偶校验位的生成和偶校验的检测。

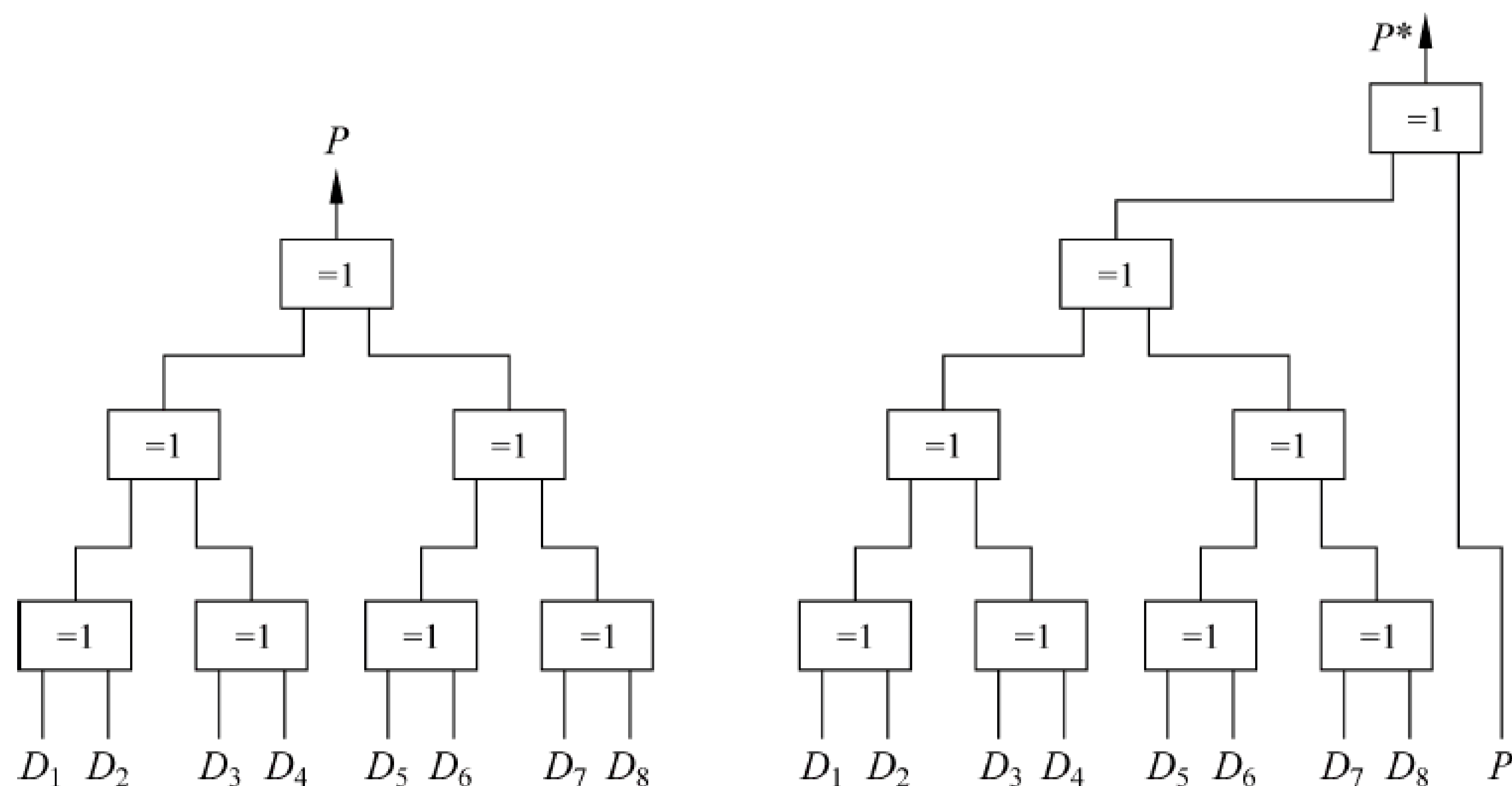


图2-18　偶校验位的生成和偶校验的检测

2.6.2　海明校验码

海明校验码是由Richard Hamming于1950年提出的，属于纠错码，它不仅具有检错能力，而且能进行错误定位，从而纠正错误。能进行错误定位意味着能知道是哪位发生了错误，对二进制而言，只需将发生错误的位变反即可纠正该位的错误。

海明校验码的基本思想是：对原数据码按某种规律分成若干组，每组安排一个校验位，进行奇偶校验，这样就可产生多位检错信息，用其不同的二进制组合，指出错误的所在，从而达到纠错的能力。这里介绍一种具有纠正1位错能力的海明码的编码方法及实现。

设有效数据位为 k 位，现设置校验位为 r 位。r 位二进制共有 2^r 个不同组合(码点)，若

要求这 2^r 个不同组合能够分别用于指出 $k+r$ 个数据位和校验位具体是哪一位出错，则应满足关系：

$$2^r \geqslant k+r+1 \tag{2.25}$$

其中还有一个码点用于指出“无错”。

按式(2.25)，可以得到数据位 k 与校验位 r 的对应关系如表 2-7 所示。

表 2-7 数据位 k 与校验位 r 的对应关系

数据位 k 值	校验位 r 值	数据位 k 值	校验位 r 值
1	2	12～26	5
2～4	3	27～57	6
5～11	4		

设 $m=k+r$，即数据码和校验码共 m 位，它们共同组成海明码 $H_m H_{m-1} \cdots H_2 H_1$，则数据码和校验码的编码规则如下：

(1) 位置安排。按以下规则进行：每个校验位 P_i 在海明码中被安排在位号为 2^{i-1} 的位置，其余各位为数据位，按从低到高逐位排列。

(2) 编码。海明码的每一位码 H_i(包括数据位和校验位)由多个校验位校验，其关系是被校验的每一位位号要等于校验它的各校验位的位号之和。

按此原则，实际是将海明码校验分成 r 组的奇偶校验，每个校验位 P_i 组成一组奇偶校验，按奇偶校验的方法进行编码，因此，海明校验码又称为分组奇偶校验码。

下面以一种(7,4)海明码为例介绍海明码的编码及检纠错方法。

所谓(7,4)海明码是指其数据位 4 位，校验位 3 位，数据编码共 7 位。设 4 位数据位为 $D_1D_2D_3D_4$、3 位校验位为 $P_1P_2P_3$，它们组成的 7 位海明码为 $H_7H_6H_5H_4H_3H_2H_1$。根据上述海明码的编码规则，3 位校验位 P_1、P_2、P_3 分别安排在 2^0、2^1 和 2^2 的位置，即对应 H_1、H_2 和 H_4，4 位数据位 D_1、D_2、D_3、D_4 则对应剩下的 H_3、H_4、H_6 和 H_7，也即

$$H_7H_6H_5H_4H_3H_2H_1 = D_4D_3D_2P_3D_1P_2P_1$$

据此，可以列出海明码位号与校验位位号的关系如表 2-8 所示。

表 2-8 海明码位号与校验位位号的关系

海明码位号	数据位/校验位	参与校验的校验位位号	被校验位的海明码位号 ＝校验位位号之和
H_1	P_1	1	1＝1
H_2	P_2	2	2＝2
H_3	D_1	1,2	3＝1＋2
H_4	P_3	4	4＝4
H_5	D_2	1,4	5＝1＋4
H_6	D_3	2,4	6＝2＋4
H_7	D_4	1,2,4	7＝1＋2＋4

若采用偶校验，则根据表 2-8，可得出 3 位校验位 P_1、P_2、P_3 的偶校验关系式如下：

$$\begin{aligned} P_1 &= D_1 \oplus D_2 \oplus D_4 \\ P_2 &= D_1 \oplus D_3 \oplus D_4 \\ P_3 &= D_2 \oplus D_3 \oplus D_4 \end{aligned} \tag{2.26}$$

三个偶校验式将对应得到三个偶校验检测位，假设记为 S_1、S_2 和 S_3，它们的求解表达式为

$$\begin{aligned} S_1 &= P_1 \oplus D_1 \oplus D_2 \oplus D_4 \\ S_2 &= P_2 \oplus D_1 \oplus D_3 \oplus D_4 \\ S_3 &= P_3 \oplus D_2 \oplus D_3 \oplus D_4 \end{aligned} \tag{2.27}$$

这样，根据 S_1、S_2 和 S_3 的状态就可以检测出是否有错，以及确定错误的位置，如表 2-9 所示。

表 2-9　错误检测表

$S_3\ S_2\ S_1$	错误位置	$S_3\ S_2\ S_1$	错误位置
000	无错	100	H_4 错
001	H_1 错	101	H_5 错
010	H_2 错	110	H_6 错
011	H_3 错	111	H_7 错

例如，假设 H_2 发生了错误，其余各位均无错，由于 $H_2=P_2$，则根据式(2.27)，

$$\begin{aligned} S_1 &= P_1 \oplus D_1 \oplus D_2 \oplus D_4 = 0 \\ S_2 &= P_2 \oplus D_1 \oplus D_3 \oplus D_4 = 1 \\ S_3 &= P_3 \oplus D_2 \oplus D_3 \oplus D_4 = 0 \end{aligned}$$

$S_3S_2S_1=010$。

(7,4)海明码能够检测和纠正 1 位错的情况，却无法检测和纠正多位错的情况。一些其他的海明码则具有更强的检纠错能力。

(7,4)海明码的硬件实现如图 2-19 所示。

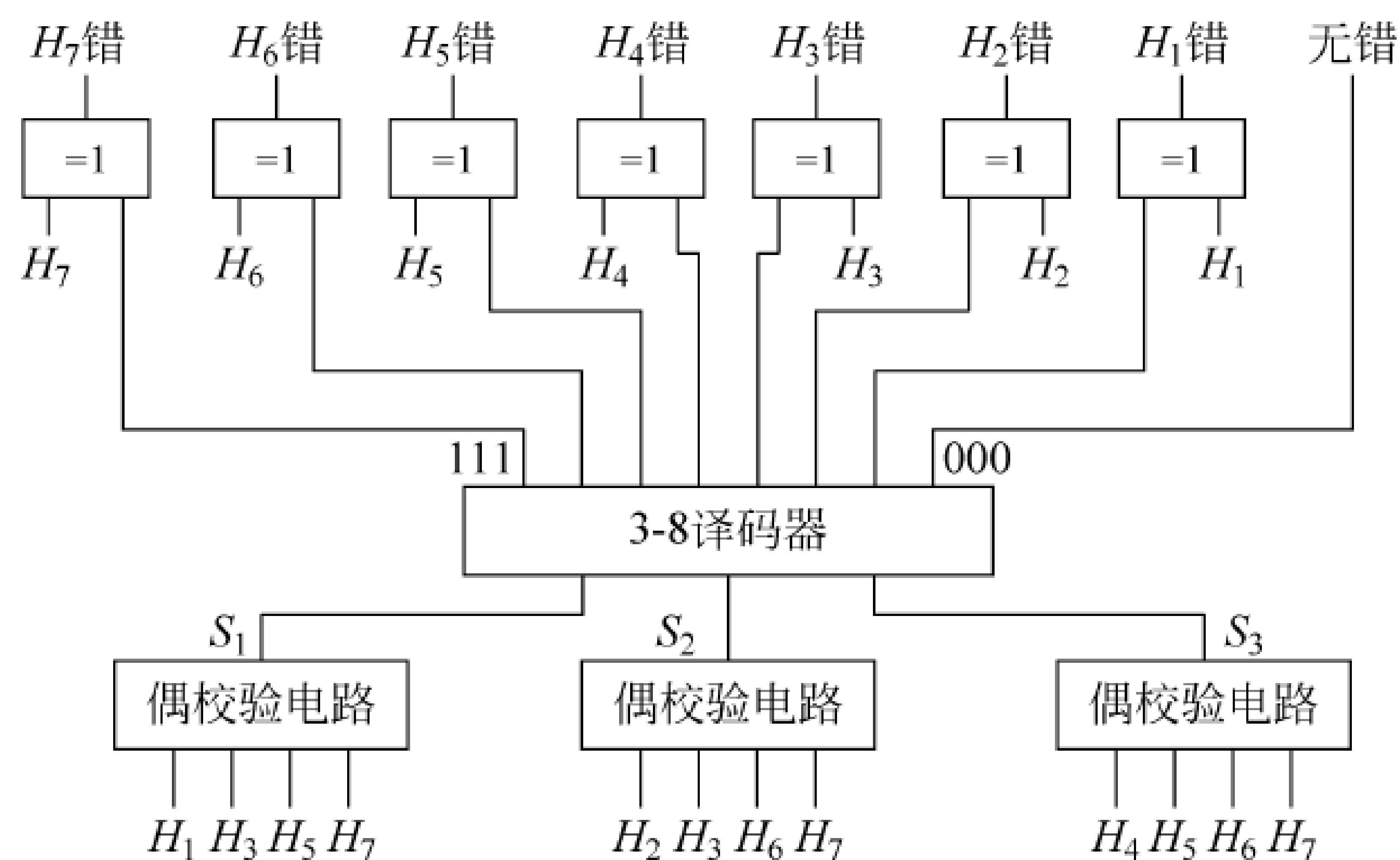


图 2-19　(7,4)海明码的硬件实现

2.6.3 循环冗余校验码

循环冗余码(Cyclic Redundancy Check,CRC)是一种具有较强检纠错能力的校验码,可以对大块数据进行校验,主要应用于主机与成组数据交换的设备之间的数据校验。例如,主机与磁盘之间的数据交换是以块(又称扇区)为单位的,一个数据块的记录格式如图 2-20 所示。

头隙	序标	数据(512/1024B)	校验	尾隙

图 2-20 主机与磁盘交换的数据块格式

这其中的校验字段通常采用的就是 CRC 循环冗余校验码。

1. 循环冗余码的编码

任何一个二进制序列都可以用一个唯一的多项式表示,例如:8 位的二进制数 10110010 可表示为 $x^7+x^5+x^4+x$。一般地,一个 k 位二进制数可唯一地表示为

$$M(x)=C_{k-1}x^{k-1}+C_{k-2}x^{k-2}+\cdots+C_1x+C_0 \tag{2.28}$$

$M(x)$称为该二进制序列的多项式,$C_i(i=0\sim k-1)$为 0 或 1。

CRC 循环冗余码的编码和检测基于的是这种二进制多项式的模 2 运算。所谓模 2 运算是指以按位模 2 加为基础的四则运算,运算时不考虑进位和借位。例如:

```
模2加          模2减          模2乘          模2除

 10011101       01010011        1101                 101
+11001010      -10011101      × 1010         1010 )100010
---------      ---------      -------              1010
 01010111       11001110         0000              ------
                                1101                0101
                               0000                 0000
                              1101                 ------
                              -------               1010
                              1110010               1010
                                                   ------
                                                       0
```

循环冗余码的编码步骤如下:

(1) 设 k 位数据位的多项式为 $M(x)$,将其左移 r 位,则得到一个 $n=k+r$ 位的多项式 $M(x)\cdot x^r$,其中,前 k 位为有效的数据位,而后 r 位将是要产生的校验位。

(2) 将 $M(x)\cdot x^r$ 除以一个 $r+1$ 位的 $G(x)$(称为生成多项式),得到一个商 $Q(x)$和一个 r 位的余数 $R(x)$,即

$$M(x)\cdot x^r=Q(x)G(x)+R(x) \quad (\text{模 2 运算})$$

则 $R(x)$即为 $M(x)$所对应校验位的多项式。

(3) 多项式 $M(x)\cdot x^r+R(x)=T(x)$即为发送方得到的 $n=k+r$ 位 CRC 校验编码。

【例 2.27】 使用生成多项式 x^3+x+1 对 4 位数据位 1100 进行 CRC 编码。

解:数据位 1100 对应的多项式为

$$M(x)=x^3+x^2 \quad (k=4)$$

将 1100 左移 3 位得到的 1100000 对应的多项式为

$$M(x)\cdot x^3=x^6+x^5 \quad (r=3)$$

$$(M(x)\cdot x^3)/G(x)=1100000/1011=1110(\text{商})+010(\text{余数})$$

$$M(x)\cdot x^3+R(x)=1100010$$

因此，4 位数据位 1100 对应的 CRC 编码为 1100010。

2. 循环冗余码的检测

下面考察一下循环冗余码的检测。接收方收到的 CRC 校验码对应的多项式为

$$\begin{aligned}T(x)&=M(x)x^r+R(x)\\&=Q(x)G(x)+R(x)+R(x)\\&=Q(x)G(x)\quad(\text{模 2 运算})\end{aligned}$$

在不出错的情况下，$T(x)$可以被 $G(x)$整除。

因此，循环冗余码的检测方法是：将接收到的校验码对应的多项式 $T(x)=M(x)\cdot x^r+R(x)$除以生成多项式 $G(x)$，若除尽，则说明码字无错；若除不尽，则说明码字有错。

生成多项式的选择对于 CRC 校验码来说非常关键，它直接影响到 CRC 码的检纠错能力，人们经过研究探索，已找出很多有效的生成多项式，其中已列为国际标准的如

CRC-CCITT：$x^{16}+x^{12}+x^5+1$

CRC-16：$x^{16}+x^{15}+x^2+1$

CRC-12：$x^{12}+x^{11}+x^3+x^2+x+1$

本章小结

本章主要讲述了以下内容：

（1）进位记数制及数制的转换。在生活中使用的是十进制，而在计算机中使用的是二进制表示数据，另外为了书写方便，常使用十六进制。任何一个 K 进制数都可以使用一个唯一对应的多项式表示，即$(N)_K=D_mK^m+D_{m-1}K^{m-1}+\cdots+D_1K^1+D_0K^0+D_{-1}K^{-1}+D_{-2}K^{-2}+\cdots+D_{-n}K^{-n}$。

（2）数值数据的机器表示。先介绍了数值数据在机器中的原、补、反、移码表示方法，然后主要介绍了计算机中的定点数表示和浮点数表示，而浮点数表示主要介绍的是 IEEE 754 标准。

（3）非数值数据的机器表示。介绍了几种常见的非数值数据表示，包括十进制数串、字符(串)及汉字等。字符的表示除了最常用的 ASCII 码外，还有其他一些编码方式，在不同场合有不同的应用。

（4）定点数和浮点数的运算。介绍了它们的运算方法和硬件实现，不同的运算方法在很大程度上影响到硬件实现的效率。

（5）数据校验码。主要讲了三种校验码：奇偶校验、海明校验和循环冗余码校验。内存校验常用奇偶校验，而主机与磁盘等外设之间的数据传输采用 CRC 校验。

习题二

一、名词解释

1. 定点数	2. 浮点数	3. 阶码	4. 尾数
5. 规格化	6. 溢出	7. 字节	8. 字
9. 舍入处理	10. 数据校验码	11. 检错码	12. 纠错码

二、选择题

1. 在机器数中，(　　)的零的表示形式是唯一的。
   A. 原码　　B. 补码　　C. 反码　　D. 原码和反码

2. 计算机系统中采用补码表示及运算的原因是(　　)。
   A. 与手工运算方式保持一致　　B. 提高运算速度
   C. 简化计算机的硬件设计　　D. 提高运算的精度

3. 设某机器采用 8 位补码定点数表示，其中符号位 1 位，数值位 7 位，则二进制 10000000 表示的十进制数为(　　)。
   A. 127　　B. 128　　C. −127　　D. −128

4. 某机器字长 32 位，采用定点整数表示，符号位为 1 位，尾数为 31 位，则可表示的最大正整数和最小负整数分别为(　　)。
   A. $+(2^{31}-1)$和$-(1-2^{-32})$　　B. $+(2^{30}-1)$和$-(1-2^{-32})$
   C. $+(2^{31}-1)$和$-(2^{31}-1)$　　D. $+(2^{30}-1)$和$-(2^{-31}-1)$

5. 在定点二进制运算器中，减法运算一般是通过(　　)来实现的。
   A. 原码运算的二进制减法器　　B. 补码运算的二进制减法器
   C. 补码运算的十进制加法器　　D. 补码运算的二进制加法器

6. 在浮点加减运算的对阶中，遵循小阶对大阶的原因是(　　)。
   A. 损失的精度小　　B. 损失的位数少
   C. 不容易产生溢出　　D. 都不是

7. 若 $x=103$，$y=-25$，则下列表达式采用 8 位定点补码运算实现时，会发生溢出的是(　　)。(2014 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)
   A. $x+y$　　B. $-x+y$　　C. $x-y$　　D. $-x-y$

8. float 型整数据常用 IEEE 754 单精度浮点格式表示，假设两个 float 型变量 x 和 y 分别在 32 位寄存器 f1 和 f2 中，若(f1)＝CC900000H，(f2)＝B0C00000H，则 x 和 y 之间的关系为(　　)。(2014 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)
   A. $x<y$ 且符号相同　　B. $x<y$ 且符号不同
   C. $x>y$ 且符号相同　　D. $x>y$ 且符号不同

9. 某字长为 8 位的计算机中，已知整型变量 x、y 的机器数分别为$[x]_{补}=11110100$，$[y]_{补}=10110000$。若整型变量 $z=2*x+y/2$，则 z 的机器数为(　　)。(2013 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)
   A. 11000000　　B. 00100100
   C. 10101010　　D. 溢出

10. 用海明码对长度为8位的数据进行检/纠错时，若能纠正一位错。则校验位数至少为(　　)。(2013年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

A. 2　　B. 3　　C. 4　　D. 5

11. 假定编译器规定 int 和 short 类型长度占32位和16位，执行下列C语言语句

```
unsigned short x = 65530;
unsigned int y = x;
```

得到 y 的机器数为(　　)。(2012年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

A. 0000 7FFA　　B. 0000 FFFA

C. FFFF 7FFA　　D. FFFF FFFA

三、综合题

1. 写出以下不同进制数据的按权展开的多项式和形式。

(1) $(10110101)_2$　(2) $(258.66)_{10}$　(3) $(D59.3C)_{16}$　(4) $(1101.011)_2$

2. 将以下十进制数分别转换成二进制数和十六进制数。

(1) 38　(2) 128　(3) 0.125　(4) 250.5

3. 将以下二进制数转换成十进制数。

(1) 10110101　(2) 11011110　(3) 0.1101　(4) 1001.0111

4. 将以下二进制数转换成十六进制数。

(1) 10101001　(2) 1000110　(3) 1101.0101　(4) 1011100.11

5. 将以下十六进制数分别转换成二进制数和十进制数。

(1) 3AC　(2) 0.12D　(3) 258B.CE　(4) FFFF

6. 一个围棋盘共有18×18=324个小方格，现假设在第一个小方格中放入1粒米，第二个小方格中放入2粒米，第三个小方格中放入4粒米，第四个小方格中放入8粒米，……，如此下去，请问：

(1) 在最后一个小方格中将放入多少粒米？

(2) 假设1两米有1万粒，那么最后一个小方格将放入多少斤米？

7. 写出以下二进制数的原码、补码、反码和移码表示。

(1) $(10110101)_2$　(2) $(-10011011)_2$　(3) $(0.11101111)_2$

(4) $(-0.00110101)_2$　(5) $(1001.1011)_2$　(6) $(-0.00101101)_2$

8. 浮点数对阶的原则是什么？

9. 浮点数加减运算包括哪些步骤？

10. 浮点数乘除运算包括哪些步骤？

11. 设机器字长为16位，浮点表示时，阶码6位，尾数10位，其中阶符和数符各1位。试问：

(1) 该浮点数的最大和最小浮点数分别为多少？

(2) 若采用规格化表示，则该浮点数的表示范围是怎样？

12. 将十进制数328.125转换成IEEE 754标准的单精度浮点数的二进制存储格式。

13. 若浮点数 x 的IEEE 754标准存储格式为$(A2E80000)_{16}$，求其浮点数的十进制值。

14. 写出以下十进制数的压缩 BCD 码表示。

(1) +125　　(2) −328　　(3) +1234　　(4) −5678

15. 求以下 x、y 的补码的加法和减法运算，并判断结果是否产生了溢出。

(1) $x=0.110110, y=0.110101$　　(2) $x=0.100101, y=-0.101101$

(3) $x=-0.000011, y=0.101001$　　(4) $x=-0.100101, y=-0.111101$

16. 设两浮点数的 IEEE 754 标准存储格式分别为

x=0 10011010 01001101000000000000000，y=1 10010100 11011011000000000000000，

试作以下计算，并给出结果的 IEEE 754 标准存储格式。

(1) $x+y$　　(2) $x-y$　　(3) $x\times y$　　(4) x/y

17. 图 2-21 为某 ALU 部件的内部逻辑图，图中 S_0、S_1 为功能选择控制线，C_{in} 为最低位的进位输入，$A(A_1\sim A_4)$ 和 $B(B_1\sim B_4)$ 是参与运算的两个数，$F(F_1\sim F_4)$ 为输出结果。试分析在 S_0、S_1、C_{in} 各种组合的条件下，输出 F 和输入 A、B、C_{in} 的算术运算关系。

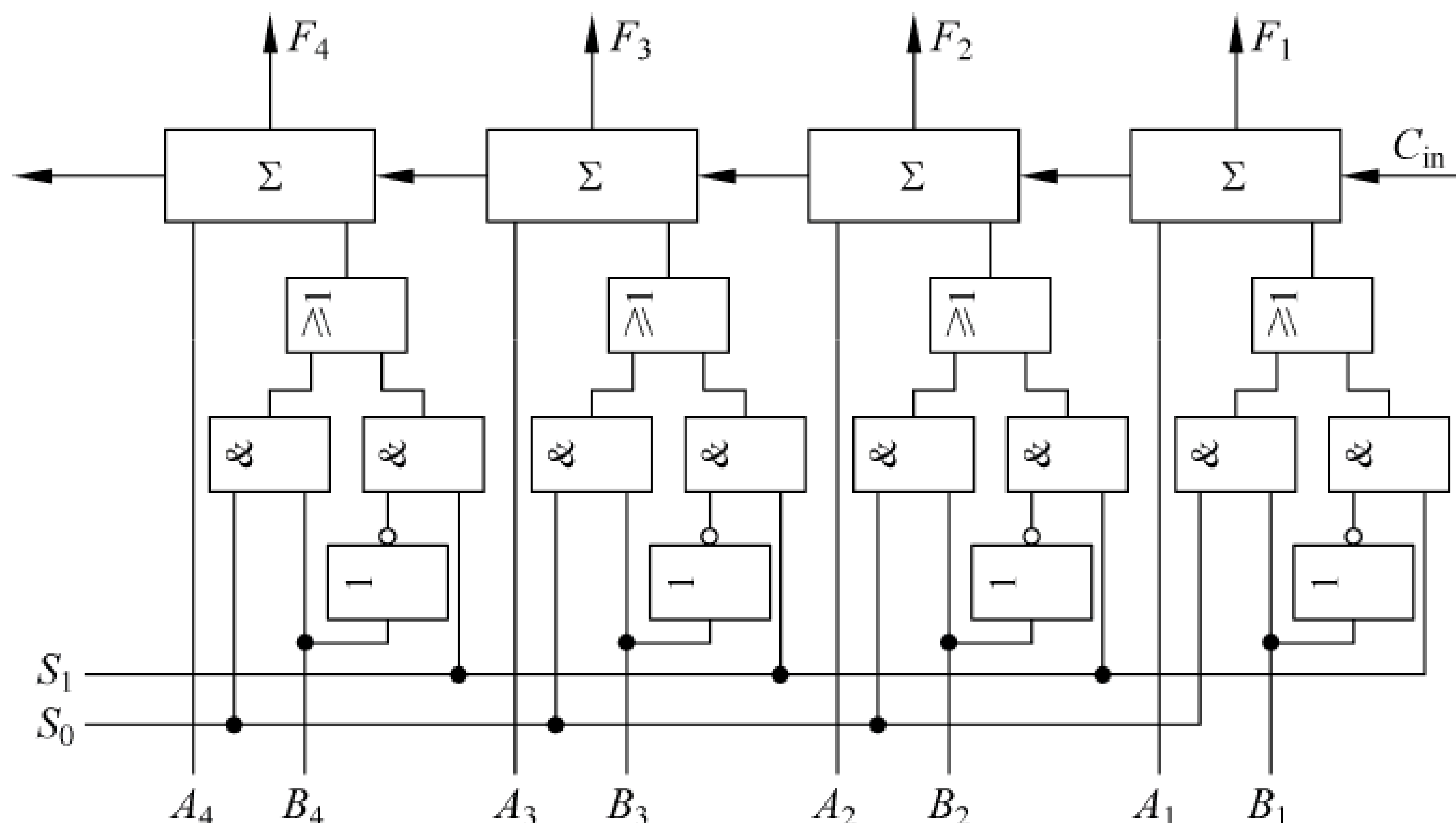


图 2-21　ALU 部件的内部逻辑图

18. (2011 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

假定在一个 8 位字长的计算机中运行如下类 C 程序段：

```
unsigned int x = 134;
unsigned int y = 246;
int m = x;
int n = y;
unsigned int z1 = x - y;
unsigned int z2 = x + y;
int k1 = m - n;
int k2 = m + n;
```

若编译器编译时将 8 个 8 位寄存器 R1～R8 分别分配至变量 x、y、m、n、$z1$、$z2$、$k1$ 和 $k2$，请回答下列问题。(提示：带符号整数用补码表示。)

(1) 执行上述程序段后，寄存器 R1、R5 和 R6 的内容分别是什么？(用十六进制表示)

(2) 执行上述程序段后，变量 m 和 $k1$ 的值分别是多少？(用十进制表示)

(3) 上述程序段涉及带符号整数加/减、无符号整数加/减运算，这四种运算能否利用同一个加法器及辅助电路实现？简述理由。

(4) 计算机内部如何判断带符号整数加/减运算的结果是否发生溢出？上述程序段中，哪些带符号整数运算语句的执行结果会发生溢出？

19. 试画出一个将4位原码转换为补码的逻辑电路图。

20. 什么情况下选择奇偶校验码，什么情况下选择海明码？

21. 设有16位数据位，如果采用海明校验，至少需要设置多少位校验位？数据位和校验位的安排是怎样的？

22. 设有8位数据位，现采用海明校验码进行编码，试说明：

(1) 需要多少位检验位？

(2) 对含有8位数据位的海明校验码进行编码。

(3) 如何进行差错检测？

(4) 画出海明码差错检测的硬件实现图。

23. 使用生成多项式 x^3+x+1 对4位数据位0101进行CRC编码。

24. 生成多项式的选择对于CRC校验码来说非常关键，它直接影响到CRC码的检纠错能力，查阅资料，进一步了解有关生成多项式的性质与检纠错能力的关系。

第3章 汇编级机器组织

现代计算机是按人们预先编制的程序进行工作的,而程序是由指令组成的。计算机能直接识别、执行的命令称为机器指令(简称指令),机器指令用二进制编码表示。一条指令规定计算机完成某种基本操作,一台计算机能执行的所有指令的集合称为该计算机的指令系统(或称指令集)。

指令系统是计算机系统性能的集中体现,其功能与格式不仅直接影响到计算机硬件结构的设计,而且与系统软件的设计、计算机的应用领域等息息相关。通常性能较好的计算机都设置有功能齐全、通用性强、指令丰富的指令系统,而指令功能的实现又需要复杂的硬件结构来支持。

虽然机器指令能被计算机直接识别和执行,但是难于书写、记忆,出错不易发现;汇编指令是机器指令的助记符表示形式,与机器指令是一一对应的。因此,本章主要讨论计算机中汇编级指令格式、地址结构、指令及操作数的寻址方式、指令种类和功能及典型指令系统举例等,还对精简指令集系统作了介绍。

3.1 汇编级机器指令系统

计算机在汇编级机器指令系统的设计上,既要考虑指令系统功能的需要,又要考虑实现其所需机器硬件的成本。但不管怎样,对机器的指令系统和指令种类有一些基本的要求。

3.1.1 指令系统的发展

在现代计算机的发展过程中,计算机的设计、编程语言、制造工艺技术都发生了深刻的变革。伴随着这种变革,计算机指令系统的发展经历了从简单到复杂的演变过程。

20 世纪 50—60 年代是现代计算机发展的早期,采用电子管或晶体管等分立元件组成计算机硬件,其体积庞大,价格也很昂贵,因此计算机的硬件结构比较简单,所支持的指令系统也只有定点加减、逻辑运算、数据传送、转移等十几至几十条最基本的指令,而且寻址方式简单。

到了 20 世纪 60 年代中期,随着集成电路的出现,计算机的体积减小、功耗下降、价格不断下降,硬件功能不断增强,指令系统也越来越丰富。除最基本的指令外,增加了乘除运算、浮点运算、十进制运算、字符串处理等指令,指令数目有一二百条,寻址方式也趋于多样化。

随着集成电路的发展和计算机应用领域的不断扩大,20 世纪 60 年代后期开始出现系

列计算机。系列计算机是指基本指令系统相同、基本体系结构相同的一系列计算机，例如Pentium系列微型计算机。一个系列往往有多种型号，但由于推出的时间不同，采用的器件不同，它们在结构和性能上有所差异。通常是新推出机种的指令系统一定包含所有旧机种的全部指令，旧机种上运行的各种软件可以不加任何修改便可在新机种上运行，大大减少了软件开发费用。系列计算机不仅较好地解决了各机种的软件兼容问题，而且新机种的性能价格比优于旧机种。

20世纪70年代，高级语言已成为大、中、小型机的主要程序设计语言，计算机应用日益普及。但是复杂的软件系统设计一直没有很好的理论指导，导致软件质量无法保证，从而出现了所谓的"软件危机"。人们认为，缩小机器指令系统与高级语言语义的差距，为高级语言提供更多的支持，是缓解软件危机有效和可行的办法。计算机设计者们利用当时已经成熟的微程序技术和飞速发展的超大规模集成电路（VLSI）技术，增设多种复杂的、面向高级语言的指令，使指令系统越来越复杂化，大多数计算机的指令系统多达几百条，由此就出现了复杂指令系统计算机（Complex Set Instruction Computer，CISC）。

CISC指令系统的主要特点是：指令格式多、寻址方式复杂、指令数量多、不同指令的使用频率相差很悬殊。大量的统计数字表明，大约有20%的指令使用频率比较高，占用了80%的处理机时间。换句话说，占指令系统80%的指令只在20%的处理机运行时间内才被用到。

由于CISC庞大的指令系统不但使计算机的研制周期变长，正确性难以保证，不易调试和维护，而且由于采用了大量使用频率低的指令而造成硬件资源浪费。计算机设计者尝试从另一条途径来支持高级语言及适应VLSI技术特点。1975年IBM公司John Cocke提出了精简指令系统计算机（Reduced Instruction Set Computers，RISC）的设想。有关RISC的内容将在后续的3.4节详细介绍。

20世纪80年代后期以来，RISC微处理器迅速发展，广泛采用了超标量和超流水线等指令级并行处理技术。但是人们又发现RISC指令系统并不能充分实现指令级并行处理，从而影响了计算机性能的进一步提高。1983年，美国J. Fisher教授提出了超长指令字（VLIW）体系结构。VLIW指令系统的设计思想是增强指令的并行性，机器指令字具有固定的格式（一种或者多种），每条指令中包含多个独立的字段，字段中的操作码被送往不同的功能部件。这是受微程序设计技术中水平型微指令的启示。

总之，计算机指令系统的发展经历了从简单到复杂，然后又从复杂到简单的演变过程。

3.1.2 指令系统性能的要求

指令系统的性能决定了计算机的基本功能，它的设计直接关系到计算机的硬件结构和用户的需要。由于计算机性能、体系结构以及使用环境等要求不同，指令系统间的差异也是很大的。同时，随着计算机技术的迅速发展，对计算机性能要求越来越高，硬件价格迅速下降，指令系统也趋向强功能、高效、复杂化。因此，试图给计算机指令系统确定一个统一的衡量标准是很困难的。一般情况下，指令系统应满足完备性、有效性、规整性和兼容性。

1. 指令系统的完备性

完备性是指在一个有限可用的存储空间，对于任何可解的问题，编写计算程序时，指令

系统所提供的指令足够使用。一般来说，为使程序高效运行和便于硬件实现，一个完备的指令系统应包括传送类指令、算术运算类指令、逻辑运算类指令、程序控制类指令、字符串指令、特权指令和调试指令。若采用统一编址，则用传送指令即可实现输入输出的目的；若采用单独编址，则需要专门的输入输出指令。

以上几种类型指令并不是互相独立的，在一定条件下可以互相替代。

2. 指令系统的有效性

有效性是指使用该指令系统的指令编制程序能高效率地运行。这个高效率表现在解题速度快、占用存储空间小。有效性是针对整个指令系统而言的，这是一个很复杂的问题，很难确定一个统一标准。有效性反映了指令的功能要求，也反映了指令系统的完备性要求。一个更完备的指令系统就会有更好的有效性，如指令系统中有多种移位指令和字符串处理指令，对于数据处理就会有较高的有效性。在新一代计算机指令系统中有专门的编辑指令，甚至某些常用的高级语言的语句也用一条指令来实现等，采取这些措施无疑将大大提高指令系统的有效性。

3. 指令系统的规整性

规整性细分为指令操作的对称性和匀齐性，指令格式与数据格式的一致性。

指令操作的对称性是指在运算时，所有寄存器(存储)单元都可以同等对待，不论哪一个操作数或运算结果都可以不受约束地存入任一单元。如传送指令，既有 A←B，也有 B←A；加法指令既有 A←A+B，也有 B←A+B 等。这种操作的对称性对于提高软件效率和使用方便是很有利的，但是，目前许多机器还不能很好地实现这一对称性。VAX-11 机的指令系统的指令操作具有较好的对称性。

指令操作的匀齐性是指一种性质的操作可适用于各种数据类型。如 VAX-11 机在数据传送、数据交换、数据测试和计算等操作时，可以匀齐地适用于三种整数数据类型(字节、字和双字)和两种浮点数据类型(短浮点和长浮点)中的每一种。指令操作的匀齐性可使汇编程序设计与编译程序无须依赖数据类型去选用指令，缩短了程序代码长度，加快了程序执行速度。匀齐性在以前的一些机器中没有很好地实现。如 IBM 370 机有字(32 位)加法，也有半字(16 位)加法；但只有字除法，却没有半字除法等。随着硬件价格进一步下降，指令操作的匀齐性有了很大的改善。

指令格式与数据格式的一致性是指令字长与数据字长有一个规整的关系，便于程序的加工处理。如机器基本字长为 32 位，长指令选 32 位，短指令选 16 位，在此字长的条件下，可以选择指令的地址格式。

4. 指令系统的兼容性

从计算机的发展过程已经看到，由于组成计算机的基本硬件发展迅速，计算机的更新换代是很快的，这就存在软件如何跟上的问题。大家知道，一台新机器推出交付使用时，仅有少量软件可提交用户，大量软件是不断充实的，尤其是应用软件，有相当一部分是用户在使用机器过程中不断产生的，这就是所谓的第三方提供的软件。为了缓解新机器的推出与原有应用软件能否继续使用之间的矛盾，1964 年 IBM 公司在设计 IBM 360 计算机中所采用

的系列机思想较好地解决了这一问题。从此以后，计算机公司生产的同一系列计算机尽管其硬件实现各不相同，但在指令系统、数据格式、I/O 系统等保持相同，因而软件完全兼容（在此基础上，产生了兼容机）。当研制该系列计算机的新型号或高档产品时，尽管指令系统可以有较大的扩充，但仍保留原来的全部指令，保持软件"向上兼容"的特点。

系列机的各型号机器由于推出的时间不同，在结构和性能上有差异，做到所有软件都完全兼容是不可能的，只能做到"向上兼容"或"向前兼容"，即低档机运行的软件可以在高档机运行。

指令系统的兼容性使得大量已有软件产品得到继承，保护了用户的前期软件投资，减少了软件的开发费用，同时，新型号机器一出现就具有丰富的软件，有利于打开市场销路。

3.1.3　指令操作的种类

不同类型的计算机所具有的指令类型是多种多样的，其指令的数量与功能、指令格式、寻址方式、数据格式都有差别，即使是一些常用的基本指令，如算术运算指令、逻辑运算指令、转移指令等也是各不相同的。但是从指令的操作功能来考虑，一个较完善的指令系统应当包括数据传送类指令、算术运算类指令、逻辑运算类指令、程序控制类指令、输入输出类指令、系统控制类指令等。

1. 数据传送类指令

计算机的操作大部分可以归结为各类信息的传送，因此数据传送类指令是最基本的指令。数据传送指令主要用于寄存器与寄存器、寄存器与存储单元、存储单元与存储单元之间的数据传送操作。对存储器来说，数据传送包括对数据的读、写操作。数据传送时，数据从源地址传送到目的地址，而源地址中的内容保持不变。因此，计算机中的数据传送实际上是数据复制。

数据传送类指令可以是一个操作数的传送，也可以是一串操作数的传送，这取决于指令的操作要求。例如，Intel 80x86 的 MOVS 指令一次可以传送一个字节或字，而加上重复执行前缀（REP）后，一次可以把一批数据从存储器的一个区域传送到另一个区域。

数据传送指令既可以是单向的，也可以是双向的，即将源操作数与目的操作数（一个字节或一个字）相互交换位置。例如 Intel 80x86 中的 XCHG 指令，源操作数与目的操作数可互换位置，源操作数可以在寄存器或主存单元中，目的操作数则只允许在通用寄存器中。

2. 算术运算类指令

算术运算类指令主要用于定点和浮点运算。这类运算包括定点加、减、乘、除运算，浮点加、减、乘、除运算，以及自加 1、自减 1、比较等，有些机器还有十进制运算指令。绝大多数算术运算指令都会影响到状态标志位。通常的标志位有进位、溢出、全零、正负和奇偶等。

比较指令是减法指令的一个特殊变化，仍是进行两数相减的运算，但结果不回送，即不保留"差"。比较指令的功能在于不破坏原来的两个操作数，而仅设置相应的标志位，为紧跟在其后的条件转移指令提供操作的依据，以决定程序的走向。

为了实现高精度的加/减运算（双倍字长或多字长），低位字（字节）加法运算所产生的进位（或减法运算所产生的借位）都存放在进位标志中；在高位字（字节）加/减运算时，应考虑

低位字(字节)的进位(或借位)。因此指令系统中除去普通的加、减指令外,一般都设置带进位加指令和带借位减指令,例如 Intel 80x86 中的 ADC、SBB 指令。

3. 逻辑运算类指令

一般计算机都具有逻辑与、或、非、异或、移位等指令,这类指令主要用于无符号数的位操作、代码转换、判断及运算。例如 Intel 80x86 中的 AND、OR、NOT、XOR 等指令。

移位指令用来对操作数实现左移、右移和循环移位操作。移位指令可分为算术移位、逻辑移位和循环移位三类,同时它们又分别包括左移和右移两种。例如 Intel 80x86 中的 SHL/SHR、SAL/SAR、ROL/ROR、RCL/RCR 指令。

4. 程序控制类指令

程序控制类指令主要用于控制程序的执行顺序,并使程序具有测试、分析与判断的能力,因此,它是指令系统中的一个重要组成部分。这类指令主要包括转移指令、转子程序与返回指令、程序中断指令等。

1) 无条件转移指令

无条件转移指令执行时,用来改变指令的正常执行顺序,不受任何约束地将程序转移到该指令指定的地址去执行,这种操作只影响到程序计数器(PC)的内容,如 Intel 80x86 的 JMP 指令。

2) 条件转移指令

条件转移指令是指仅当满足指令规定的条件时,才执行转移;否则只相当于一条空操作指令,不改变程序执行顺序。条件转移指令一般常跟在比较或测试指令之后,根据测试条件是否满足来决定是否转移。由于这种指令可使计算机根据条件(输入数据处理或处理结果)来选择程序执行方向,因此条件转移指令使机器具有逻辑判断能力。

一般用来作为转移条件的标志有进位标志(C)、结果为零标志(Z)、结果为负标志(N)和结果溢出标志(V)等,也可以利用这些标志的组合作为转移条件,如 Intel 80x86 的 JZ、JC 等指令。

3) 转子程序与返回指令

通常,子程序是一组可以共享的指令序列,在执行主程序的过程中,可被多次调用,甚至可被不同的程序所调用。只要给出子程序的入口地址就能从主程序转入子程序。转子程序操作与转移指令之间的区别在于:转移指令无须返回,不必保存返回地址;而转子指令必须以某种方式保存返回地址,以便于子程序执行结束返回到调用程序的断点,如 Intel 80x86 的 CALL、RET 指令。

4) 程序中断指令

中断是计算机内部突发事件或由 I/O 设备请求而随机产生的。但有些计算机为了方便程序调试或调用某种功能等目的,设置有专门的程序中断指令,如 Intel 80x86 的 INT 指令。当程序运行该类指令时,就以中断方式暂停后续指令的执行,然后将程序断点参数保存,接着转向某地址执行某段程序、实现所期望的功能,如查错、显示结果等。由于这些指令是由软件驱动的,所以又称为软中断。

5. 输入输出类指令

输入输出类指令是用来启动外围设备,检查测试外围设备的工作状态,实现外围设备与主机之间,或外围设备与外围设备之间的信息交换。不同计算机的输入输出方式差别很大,通常有两种方式:独立编址方式和统一编址方式。

独立编址方式设置专用的输入输出指令,指令中应给出与存储器地址无关的设备码(或端口地址),而指令操作码规定本指令要求的输入输出操作,包括输入、输出、启动等,如Intel 80x86 的 IN、OUT 等指令。

所谓统一编址就是把输入输出设备寄存器和主存单元统一编址。在这种方式下,用一般的数据传送指令来实现输入输出操作。一个输入输出设备通常至少有三个寄存器:数据寄存器、命令寄存器和状态寄存器。每个寄存器都可以由分配给它们的唯一的主存地址来识别,主机可以像访问主存一样去访问输入输出设备的寄存器。VAX-11 机采用的就是统一编址方式,它把最高的 4KB 主存地址作为输入输出设备寄存器的地址。

6. 特权指令

特权指令是指具有特殊权限的指令,它用于多用户、多任务计算机的系统资源分配与管理。这类指令的功能包括:改变系统的工作方式,检测用户的访问权限,修改虚拟存储器管理的段表、页表,完成任务的创建和切换等。为了安全,这类指令一般不直接提供给用户使用,如 Pentium 中的 SGDT、LSI、INVD 等。

7. 处理器控制指令

处理器控制指令是直接控制 CPU 实现某种功能的指令。包括状态标志位的操作指令、停机指令、等待指令、空操作指令、封锁总线指令等,如 Intel 80x86 中的 STC、WAIT、NOP、LOCK。

知识拓展

x86CPU 指令系统的扩展指令集

目前市面上 Intel 和 AMD 的处理器在 x86 基本指令集的基础上,为了提升处理器各方面的性能,又各自开发新的指令集。

MMX 指令集:它是 Intel 公司为 Pentium MMX 处理器所开发的多媒体指令集。MMX 指令集中包括了 57 条多媒体指令,通过这些指令可以一次性处理多个数据。使用 MMX 指令集的好处就是所使用的操作系统可以在不做任何改变的情况下执行 MMX 指令。但是,MMX 指令集不能与 x86 的浮点运算指令同时执行。

SSE 指令集:SSE 指令集叫单指令多数据流(SIMD)扩展。它最先运用于 Pentium Ⅲ处理器,可以提高处理器浮点性能,它共有 70 条指令,其中包含提高 3D 图形运算效率的 50 条 SIMD 浮点运算指令、12 条 MMX 整数运算增强指令、8 条优化内存中的连续数据块传送指令。理论上这些指令对图像处理、浮点运算、3D 运算、多媒体处理等众多多媒体的应用能力起到了全面提升的作用。SSE 指令包含了 3DNow! 指令集(3DNow! 是 AMD 公司运用于 AMD 的 K6、K6-2 和 K7 系列处理器中的指令集)的绝大部分功能,只是实现的方法不同

而已。Pentium 4 发布之后,Intel 又推出了更先进的 SSE2 指令集。SSE2 包含了 144 条指令,由 SSE 指令集和 MMX 指令集两部分组成。SSE 部分主要负责处理浮点数,而 MMX 部分则专门计算整数。SSE2 的寄存器容量是 MMX 寄存器的两倍,在指令处理速度保持不变的情况下,通过 SSE2 优化后的程序和软件运行速度也能够提高两倍。

SSE3 指令是目前规模最小的指令集,它只有 13 条指令。包括数据传输命令、数据处理命令、特殊处理命令、优化命令、超线程性能增强五个部分,SSE3 可以提升处理器的超线程的处理能力。

上面介绍的扩展指令集,对于处理器的性能提升受到 IA-32 体系的限制,x86 架构基本上不会再有具有革命性意义的指令集出现,Intel 和 AMD 都已经把重心转向了 64 位体系架构的处理器指令集开发上。

3.2 指令格式

计算机的指令格式与机器的字长、存储器的容量及指令的功能都有很大的关系。从便于程序设计、增加基本操作并行性、提高指令功能的角度来看,指令中应包含多种信息。但在有些指令中,由于部分信息可能无用,这将浪费指令所占的存储空间,并增加了访存次数,也许反而会影响速度。因此,如何合理、科学地设计指令格式,使指令既能给出足够的信息,又使其长度尽可能地与机器字长相匹配,以节省存储空间,缩短取指时间,提高机器的性能,这是指令格式设计中的一个重要问题。

计算机是通过执行指令来处理各种数据的。为了指出所执行的操作、数据的来源及操作结果的去向,一条指令必须包含下列信息。

(1) 操作码:它具体说明了操作的性质及功能。一台计算机可能有几十条至几百条指令,每一条指令都有一个相应的操作码,计算机通过识别该操作码来完成不同的操作。

(2) 操作数的地址:CPU 通过该地址就可以取得所需的操作数。

(3) 操作结果的存储地址:把对操作数的处理结果保存在该地址中,以便再次使用。

(4) 下条指令的地址:执行程序时,大多数指令按顺序依次从主存中取出执行。只有在遇到转移指令时,程序的执行顺序才会改变。为了压缩指令的长度,可以用一个程序计数器(Program Counter,PC)存放指令地址。每执行一条指令,PC 中的指令地址就自动加 1(设该指令只占一个主存单元),指出将要执行的下一条指令的地址。当遇到执行转移指令时,则用转移地址修改 PC 的内容。由于使用了 PC,指令中就不必明显地给出下一条将要执行指令的地址。

从上述分析可知,一条指令实际上包括两种信息,即操作码和地址码。因此,一条指令的结构可用如图 3-1 所示的形式来表示。

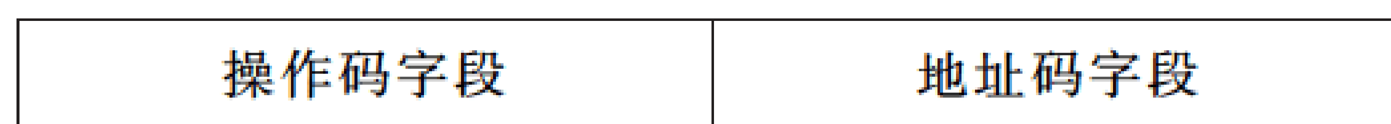


图 3-1 指令结构

3.2.1 指令字长

指令字长是指一条指令中所包含的二进制代码的位数,也就是指令的长度。

指令字长取决于操作码的长度、地址码的个数及长度。指令字长与机器字长没有固定的关系，它可以等于机器字长，也可以大于或小于机器字长。字长较短的小型、微型机中，大多数指令的长度可能大于机器字长；而在字长较长的大、中型机中，大多数指令的长度则往往小于或等于机器字长。通常，把指令字长等于机器字长的指令称为单字长指令，指令字长等于半个机器字长的指令称为半字长指令，指令字长等于两个机器字长的指令称为双字长指令。例如，IBM 370 系列机，其指令格式有 16 位(半字)的，有 32 位(单字)的，还有 48 位(一个半字)的。

在指令系统中，若指令的长度随指令功能而异，称为变长指令结构。不同的指令可以有不同的长度，即需长则长、需短则短。但因为主存一般是按字节编址的，所以指令字长多为字节的整数倍。如在 Pentium 系列机中，指令长度是可变的，有 1 字节、2 字节、4 字节、8 字节，最长的有 12 字节。原则上讲，短指令比长指令好，主要是它能节省存储空间，提高取指令的速度，但有其局限性。长指令可能会占用两个或多个地址，取指令的时间相对来说就要长些，但可以扩大寻址范围或可以带几个操作数。两者各有所长，如果长、短指令可在同一机器中混合使用，就会给系统带来很大的灵活性。

到目前为止，主流 PC 和传统的大、中、小型机仍广泛采用复杂指令系统，广泛采用变长指令格式。变长结构指令系统比较灵活，能充分利用指令字的每一位代码，但指令的控制较复杂。

在一个指令系统中，若所有指令的长度都是相等的，称为定长指令结构。定长结构指令系统控制简单，但不够灵活。为了获得更高的执行速度，采取精简指令系统 RISC，它采用定长结构指令，该技术可广泛用于工作站类的高档微机，或采用众多的 RISC 处理器构成大规模并行处理阵列。

3.2.2 地址码

根据一条指令中有几个地址码，可将该指令称为几操作数指令或几地址指令。一般的操作数有源操作数、目的操作数及操作结果这三种数，因而就形成了三地址指令格式，这是早期计算机指令的基本格式。在三地址指令格式的基础上，后来又发展成二地址格式、一地址格式和零地址格式。各种不同操作数的指令格式如图 3-2 所示。

三地址指令	OP	A_1	A_2	A_3
二地址指令	OP	A_1	A_2	
一地址指令	OP	A		
零地址指令	OP			

图 3-2 指令格式

1. 三地址指令

三地址指令字中有三个地址码，其功能为

$$A_3 \leftarrow (A_1)\ OP\ (A_2)$$

式中，OP 表示操作性质，如加、减、乘、除等；A_1、A_2、A_3 可以是内存单元的地址，也可以是运算器中通用寄存器的地址，A_1 为源操作数地址，A_2 为目的操作数地址，A_3 为存放操作结果的地址；←表示把操作结果传送到指定的地方。

这种格式的指令长度比较长，小型、微型机中很少使用。

2. 二地址指令

二地址指令常称为双操作数指令，它的两个地址码分别指明参与操作的两个操作数存放的内存单元地址或通用寄存器的名称。A_1 为源操作数地址，A_2 为目的操作数地址，A_2 兼作存放操作结果的地址。指令执行之后，A_2 地址中原来存放的内容已被覆盖了，其功能为

$$A_2 \leftarrow (A_1)\ OP\ (A_2)$$

二地址指令格式中，从操作数的物理位置来说，又可归结为三种类型。

存储器-存储器(SS)型指令：操作数都存放在内存单元中，从内存某单元中取操作数，操作结果存放至内存另一单元中，因此机器执行这种指令需要多次访问内存。

寄存器-寄存器(RR)型指令：这类指令中可使用通用寄存器组或个别专用寄存器存放操作数，从寄存器中取操作数，把操作结果放到另一寄存器。机器执行寄存器-寄存器型指令的速度很快，因为执行这类指令，不需要访问内存。

寄存器-存储器(RS)型指令：这类指令的一个操作数存放在通用寄存器组中的某个寄存器，另一个操作数存放在内存单元。执行此类指令时，既要访问内存单元，又要访问寄存器。

二地址指令是最常见的指令格式，在各种计算机的指令系统中广泛采用。

3. 一地址指令

一地址指令中只给出一个地址，该地址既是操作数的地址，又是操作结果的存储地址。如加 1、减 1 和移位等单操作数指令均采用这种格式，其操作过程是对这一地址所指定的操作数执行相应的操作后，得出的结果又存回该地址中，其功能为

$$A_1 \leftarrow OP(A_1)$$

另一种情况是以 CPU 中的累加寄存器(AC)的内容作为一个隐含操作数，指令的地址码 A 指明的是另一个操作数，操作结果又送回累加寄存器(AC)，其功能为

$$AC \leftarrow (AC)OP(A)$$

4. 零地址指令

零地址指令格式只有操作码字段。这种格式可用于只需操作码字段而不需要操作数的指令，如停机、空操作、清除等控制指令。

但对于某些需要操作数的算术、逻辑类指令也可以用零地址指令来实现，这时操作数地址是隐含的。如对累加器 AC 的内容进行操作，其 AC 为隐含约定时可用零地址。在堆栈计算机中的算术、逻辑类指令就是零地址指令，此时参加运算的操作数放在堆栈中，运算结果也放在堆栈中(有关堆栈的详细内容在后面介绍)。

5. 多地址指令

某些性能较好的大、中型机甚至高档小型机中，往往设置一些功能很强的、用于处理成

批数据的指令，如字符串处理指令、向量和矩阵运算指令等。为了描述一批数据，指令中需要多个地址来指出数据存放的首地址、长度和下标等信息。例如，CDC STAR-100 的矩阵运算指令，其地址码部分有 7 个地址段，用来指出两个矩阵的数据存放情况及结果的存放情况。

指令中的地址个数的选取要考虑诸多因素。从缩短程序长度、用户使用方便和增加操作并行度等方面来看，选用多地址指令格式较好；从缩短指令长度，减少访存次数，简化硬件设计等方面来看，一地址指令格式较好。对于同一个问题，用三地址指令编写的程序行数最少，但程序目标代码的长度最长；而用二、一、零地址指令来编写程序，程序行数一个比一个多，但程序目标代码的长度一个比一个短。表 3-1 给出了不同地址数指令的特点及适用场合。

表 3-1 不同地址数指令的特点及适用场合

地址数量	程序长度	程序存储空间	执行速度	适用场合
三地址	短	最大	一般	向量、矩阵运算为主
二地址	一般	大	较慢	广泛使用
一地址	较长	较大	较快	连续运算，硬件结构简单
零地址	较长	最小	最快	嵌套、递归问题

3.2.3 操作码

指令系统的每一条指令都有一个唯一的用二进制代码表示的操作码（Operation Code，OP），它用来表示该指令应进行什么性质的操作，如进行加法、减法、乘法、除法、数据传送等，其长度取决于指令系统中的指令条数和编码格式。

指令操作码主要有两种编码格式：固定长度格式和可变长度格式。

1. 固定长度操作码

操作码的长度固定，且集中放在指令字的一个字段中。这种格式对于简化硬件设计、减少指令译码时间非常有利，在字长较长的大、中型机和超级小型机以及 RISC 上被广泛采用。若某机器的操作码字段长度为 K 位，则它最多能表示 2^K 条不同指令。如 IBM 370 机（字长 32 位）其操作码字段都是 8 位，8 位操作码允许指定 256 条指令，而实际上在 IBM 370 机中只有 183 条指令，存在着较大的信息冗余。

IBM 370 机的指令可分为三种不同的长度形式：半字长指令、单字长指令和一个半字长指令。半字长指令不包含主存地址，单字长指令只指明一个主存地址，一个半字长指令则给出两个主存地址，共有 5 种格式，如图 3-3 所示。

【例 3.1】 机器字长为 16 位，指令格式如图 3-4 所示，其中 OP 为操作码，试分析该指令格式的特点。

解：

（1）单字长二地址指令；

（2）操作码字段 OP 为 7 位，最多能表示 128 条不同指令；X 字段用于寻址方式；

（3）源寄存器和目的寄存器都是通用寄存器（可分别指定 16 个），所以是 RR 型指令，

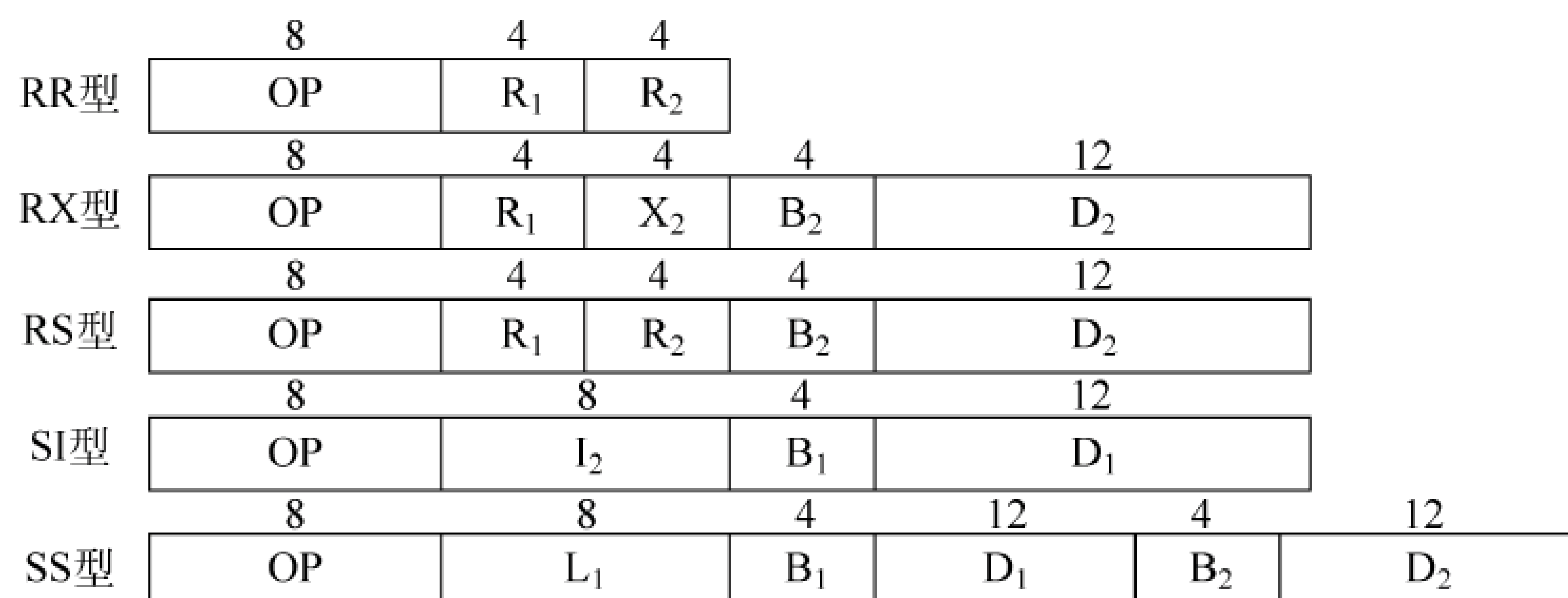


图 3-3 IBM 370 机的指令格式

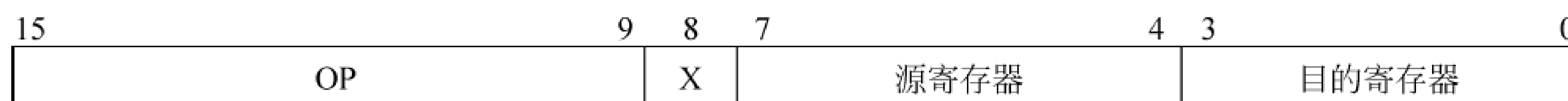


图 3-4 例 3.1 图

两个操作数均在寄存器中；

(4) 这种指令结构常用于算术逻辑运算类指令。

【例 3.2】 机器字长为 16 位，指令格式如图 3-5 所示，OP 为操作码字段，试分析指令格式特点。

图 3-5 例 3.2 图

解：

(1) 双字长二地址指令，用于访问存储器；

(2) 操作码字段 OP 为 7 位，最多能表示 128 条不同指令；X 字段用于寻址方式；

(3) 一个操作数在源寄存器(共 16 个)，另一个操作数在存储器中(由变址寄存器和位移量决定)，所以是 RS 型指令。

2. 可变长度操作码

可变长度格式操作码不但长度可变，而且分散地放在指令字的不同字段中。方法是：当指令中的地址码位数较多时，让操作码的位数少些；当指令中的地址码位数减少时，让操作码的位数增多，以增加指令种类，称之为扩展操作码。这样既能充分利用指令字的各个字段，又能在不增加指令长度的情况下扩展操作码的长度，使它能表示更多的指令。这种格式在字长较短的小型和微型机上广泛采用，如 PDP-11/VAX-11、Intel 80x86 和 Pentium 等，PDP-11 机的指令分为单字长、二字长、三字长三种，操作码字段占 4～16 位不等，可遍及整个指令长度。其部分指令格式，如图 3-6 所示。

显然，操作码长度不固定将增加指令译码和分析的难度，使控制器的设计复杂化，因此对操作码的编码至关重要。通常是在指令字中用一个固定长度的字段来表示基本操作码。

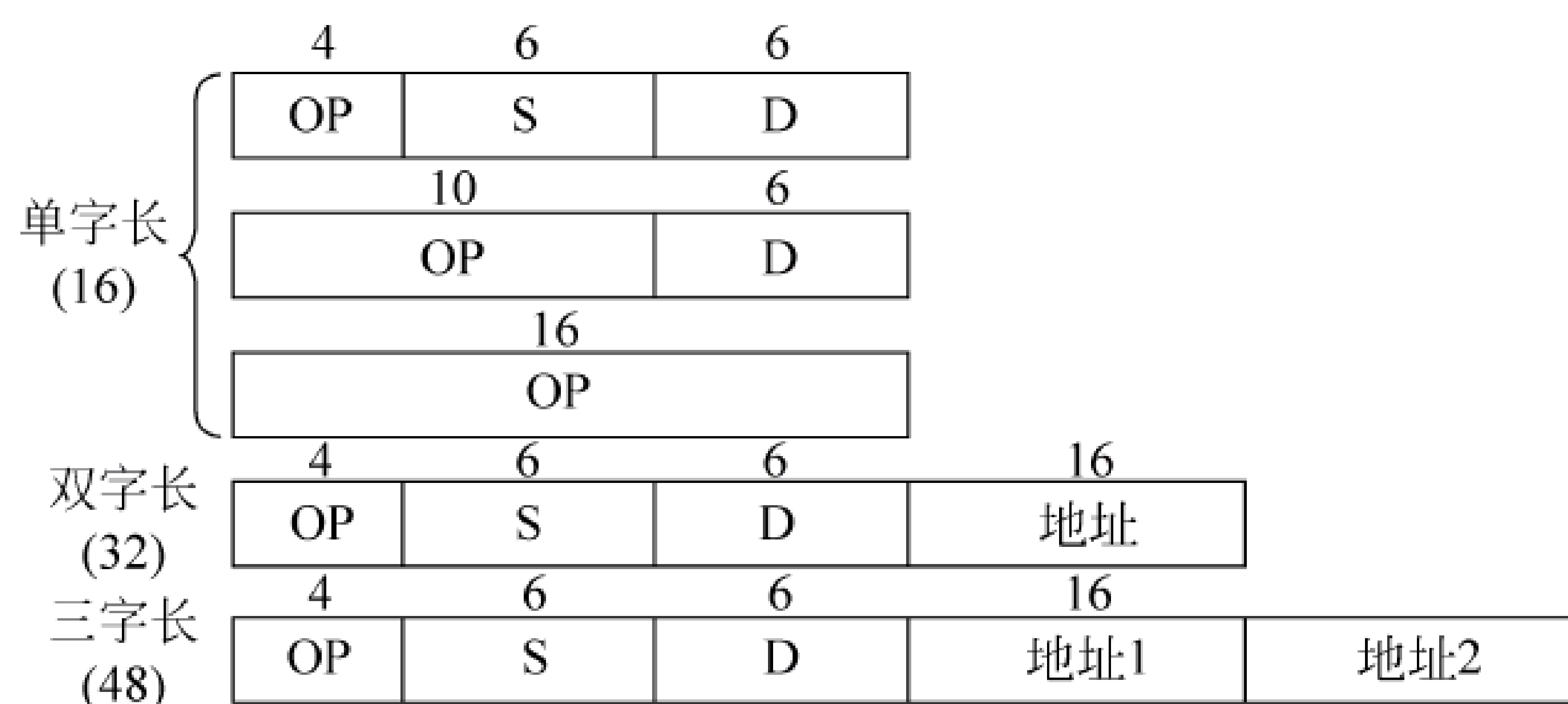


图 3-6　PDP-11 的指令格式

例如，设某机器的指令字长度为 16 位，包括一个 4 位的基本操作码字段和三个 4 位地址码字段，其格式如图 3-7 所示。

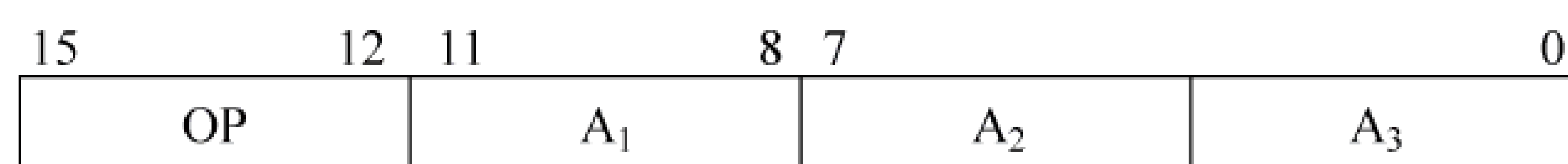


图 3-7　指令格式

4 位基本操作码有 16 种编码组合，若全部用于表示三地址指令，则有 16 条。但是，若三地址指令仅需 15 条，两地址指令需 15 条，一地址指令需 15 条，零地址指令需 16 条，共 61 条指令，应如何安排操作码？显然，只有 4 位基本操作码是不够的，必须将操作码的长度向地址码字段扩展才行。一种可供扩展的方法和步骤如下：

(1) 15 条三地址指令的操作码由 4 位基本操作码从 0000～1110 给出，剩下 1111 用于把操作码扩展到 A_1，即 4 位扩展到 8 位。

(2) 15 条二地址指令的操作码由 8 位操作码从 11110000～11111110 给出，剩下 11111111 用于把操作码扩展到 A_2，即从 8 位扩展到 12 位。

(3) 15 条一地址指令的操作码由 12 位操作码从 111111110000～111111111110 给出，剩下 111111111111 用于把操作的扩展到 A_3，即从 12 位扩展到 16 位。

(4) 16 条零地址指令的操作码由 16 位操作码从 1111111111110000～1111111111111111 给出。

一般将指令操作码放在指令字的第一字节，当读出操作码后就可马上判定：是一条单操作数指令还是一条双操作数指令，或者是零地址指令，从而知道后面还应该取几个字节指令代码。当然，在采取预取指令技术时，这个问题会复杂一些。

实际上，在可变长度操作码的指令系统设计中有一个重要的原则，就是使用频率高的指令应分配短的操作码，使用频率低的指令相应地分配较长的操作码，哈夫曼(Huffman)编码法就是根据上述原则进行编码的。这样不仅可以有效地缩短操作码在程序中的平均长度，节省存储器空间，而且缩短了使用频率高的指令的译码时间，提高了程序的运行速度。

【例 3.3】 指令字长为 12 位，每个地址码为 3 位，采用扩展操作码的方式，设计 4 条三地址指令、255 条一地址指令和 8 条零地址指令。

(1) 写出扩展表示；

(2) 画出指令译码逻辑图；

(3) 计算操作码平均长度。

解：

(1) 操作码的扩展表示如下：

$$\left.\begin{array}{c}0\,0\,0\times\times\times\times\times\times\times\times\times\\ \vdots \qquad\qquad\qquad \vdots\\ 0\,1\,1\times\times\times\times\times\times\times\times\times\end{array}\right\}\text{4 条三地址指令}$$

$$\left.\begin{array}{c}1\,0\,0\,0\,0\,0\,0\,0\,0\times\times\times\\ \vdots \qquad \vdots\\ 1\,1\,1\,1\,1\,1\,1\,1\,0\times\times\times\end{array}\right\}\text{255 条一地址指令}$$

$$\left.\begin{array}{c}1\,1\,1\,1\,1\,1\,1\,1\,1\,0\,0\,0\\ \vdots \qquad \vdots\\ 1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\end{array}\right\}\text{8 条零地址指令}$$

(2) 指令译码逻辑图如图 3-8 所示。

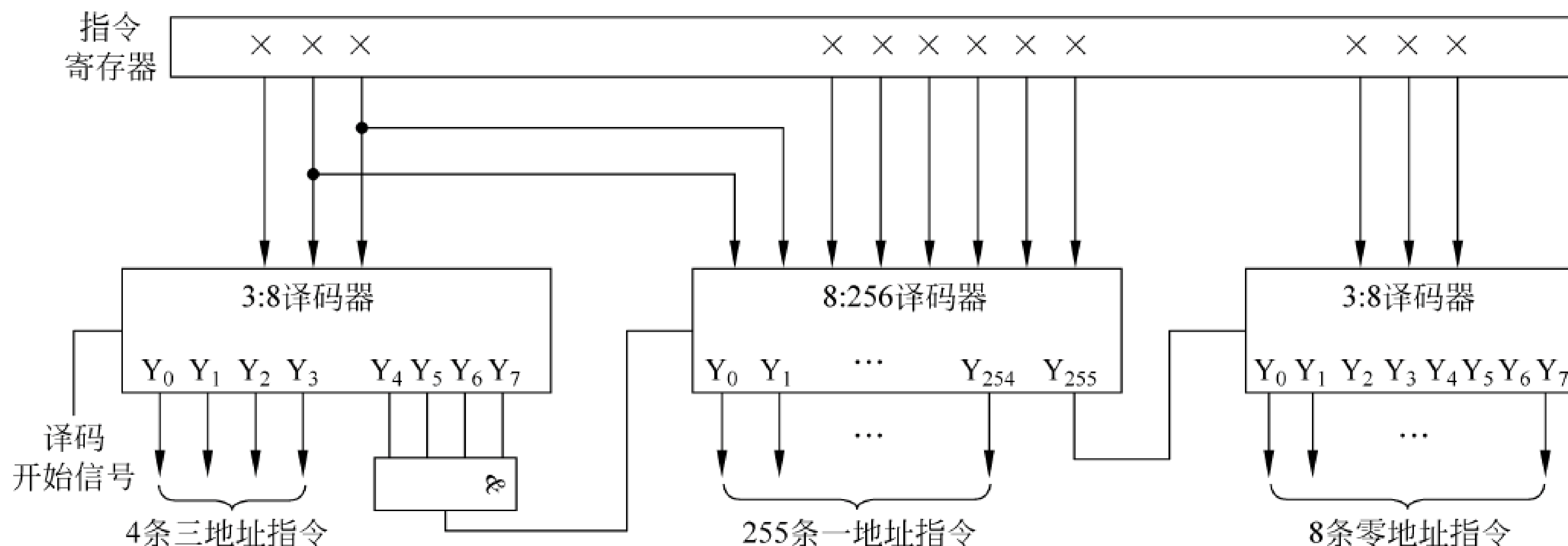


图 3-8　指令译码逻辑图

(3) 指令操作码平均长度$=(4\times3+9\times255+12\times8)/267=9$(位)。

3. 指令助记符

由于硬件只能识别 1 和 0，所以采用二进制操作码是必要的，但是用二进制代码来书写程序却非常麻烦。为了便于书写和阅读程序，每条指令通常用 3 个或 4 个英文缩写字母来表示，这种缩写码叫做指令助记符，如表 3-2 所示。这里假定指令系统只有 7 条指令，所以操作码只需 3 位二进制。

表 3-2　典型的指令助记符

典型指令	指令助记符	二进制操作码
加法	ADD	001
减法	SUB	010
数据传送	MOV	011
跳转	JMP	100
转子程序	CALL	101
存操作数	STR	110
取操作数	LDA	111

由于指令助记符提示了每条指令的意义，因此比较容易记忆，书写起来比较方便，阅读程序容易理解。例如，一条加法指令，可以用助记符 ADD 来代表操作码 001；而一条数据传送指令，可以用助记符 MOV 来代表操作码 011。

由指令助记符构成的指令称为汇编指令，讨论指令系统时为了方便，经常使用汇编指令。需要注意的是，在不同的计算机中，指令助记符的规定是不一样的，如 80x86 系列 CPU 的汇编指令与 MCS-51 单片机的汇编指令是不同的。

硬件只能识别和执行二进制语言，因此，汇编指令还必须转换成与它们相对应的二进制码。这种转换借助汇编程序可以自动完成，汇编程序相当于一个“翻译”。

3.3　数据的存储与寻址方式

存储器中存储的数据包括两大类：一是指令，二是操作数。对这些数据的存储和寻址，不同的计算机会有不同的方法。

3.3.1　数据的存储方式

1. 操作数的存储方式

机器指令可以对不同类型的操作数进行操作。操作数的类型有数值数据和非数值数据，数值数据可以是定点整数、浮点数、十进制数，操作数的位数可以是字节、字、双字、四字。

一个多字节的数据在按字节编址的主存中通常有两种排序方式：大端次序（big-endian ordering）和小端次序（little-endian ordering）。大端次序将最高有效字节存储在最小地址单元，小端次序将最低有效字节存储在最小地址单元。图 3-9 是 4 字节的十六进制数 12345678 在存储器的存储方式示意图。

地址	值
4	12
5	34
6	56
7	78

大端方式

地址	值
4	78
5	56
6	34
7	12

78

小端方式

图 3-9　多字节数据的存储方式

采用大端方式的优点：

（1）字符串排序方便，因为字符串与整型数据的字节顺序一致，在对大端次序字符串进行比较时，整型 ALU 可以同时比较多个字符。

（2）顺序的一致性，采用相同的顺序存储整型数和字符串，显示十进制数以及字符串很方便，所有的数值可直接从左到右顺序显示而不会产生混淆。

采用小端方式的优点是适合于超长数据的算术运算，对小端次序存储的数据进行高精度算术运算，不需要寻找最低字数据，然后反向移动到较高位。

Intel 80x86、Pentium、VAX 和 Alpah 是小端次序的处理器，IBM S370/390、Motorola 的 680X0、Sun SPARC 和大多数 RISC 机器采用大端次序，Power PC 是一个既支持大端次序又支持小端次序的机器。当数据由一种端序类型的机器传送到另一类型的机器时，或当程序试图对多字节数据的个别字节或个别位进行处理时，要特别注意数据的存储次序。

2. 数据对齐方式

在数据对齐存储方式下，要求一个数据字占据完整的一个字的存储位置，而不是分成两

部分各占据一个字存储位置的一部分。例如,一个32位的数据字,在按字对齐方式下它的地址应当是4的倍数,即其地址的二进制码的最低两位为00。这样,它实际占据的存储单元的地址为$4n$、$4n+1$、$4n+2$和$4n+3$(n为自然数)。在32位宽度的存储器中,因为其字地址为$4n$,这个数据可以一次读取或者写入。如果这个数据字不按对齐方式存储,其存储单元的地址出现如$4n-1$、$4n$、$4n+1$和$4n+2$的情况,这样的数据在32位的存储器需要分两次读取或者写入。在计算机中,规定数据的存储必须按字对齐的方式进行。图3-10(a)是一个字未对齐的例子,图3-10 (b)是一个字对齐的例子。所以在存储器中有些位置是空白的,这种空白存储位置是为了保证后继数据按字对齐方式存储。

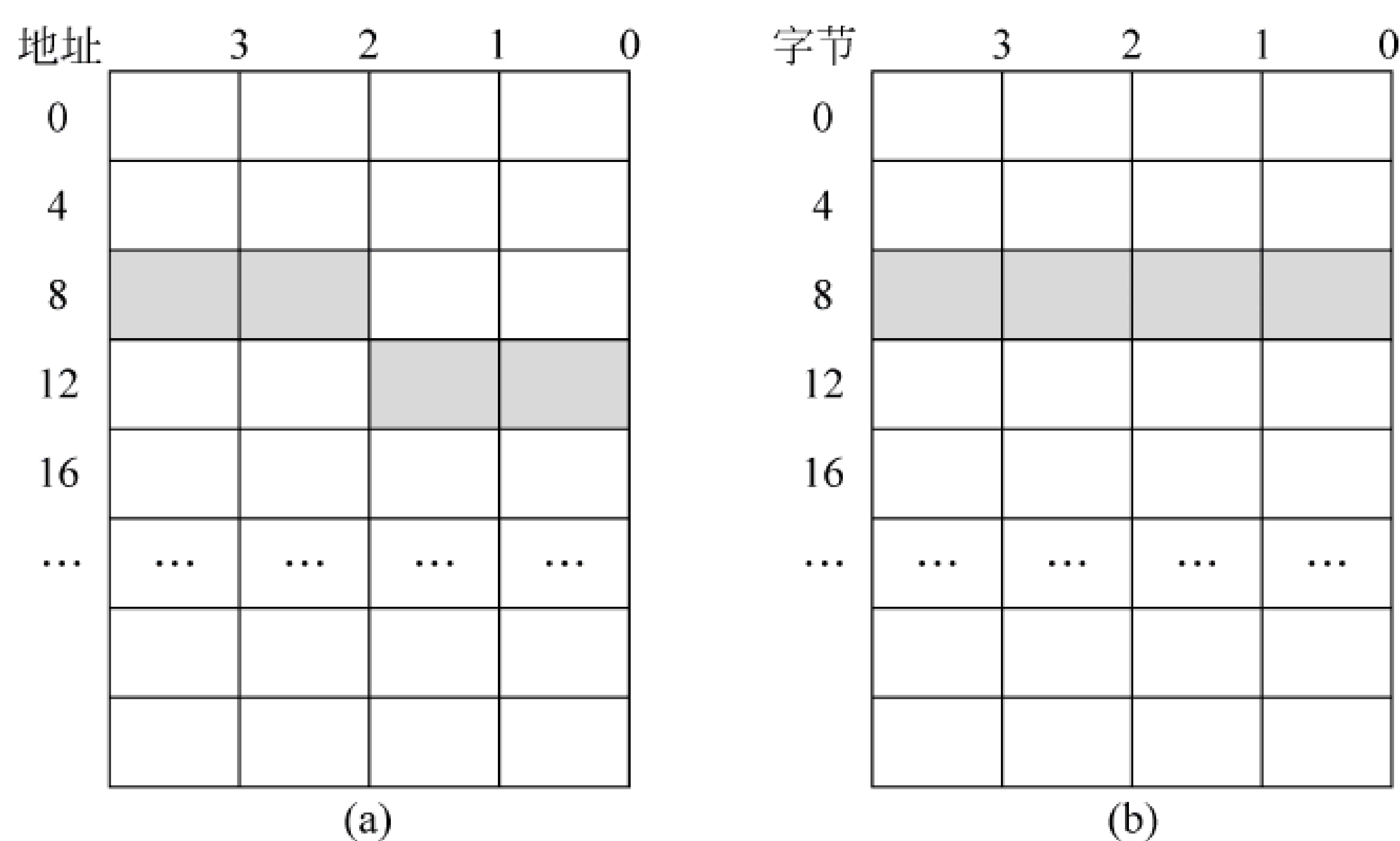


图3-10 存储器中数据字的对齐方式

3.3.2 寻址方式

计算机中有两种信息,即指令和数据(或称操作数),它们都存放在存储器相应的存储单元中。运行程序时,计算机逐条执行指令,并对数据进行处理。如何从存储器中找到所需要的指令或数据呢?很明显,只要找到它们在存储器的有效地址即可。所谓寻址方式,就是形成指令或数据有效地址的方法。

寻址方式是指令系统设计的重要内容。一套好的寻址方式能给用户提供丰富的程序设计手段,能提高程序的质量和存储空间的利用率。寻址方式是不同计算机的重要特色之一,它们对寻址方式的分类及寻址方式的名称有各自的规定。下面介绍大多数计算机都具有的基本寻址方式,有些基本寻址方式可以组合使用,从而形成更复杂的寻址方式。

1. 指令的寻址方式

指令的寻址方式分为顺序方式和跳跃方式两种。

(1) 顺序寻址方式。

组成程序的指令序列在主存中是顺序存放的。因此,执行程序是从该程序的第一条指令开始,逐条取出并逐条执行的,这种程序的顺序执行过程,称为顺序寻址方式。为了达到顺序寻址的目的,CPU中可以用一个程序计数器(Program Counter,PC)对指令的地址进行顺序计数。

PC中开始是存放程序的首地址,然后每取出一条指令,PC加1或加一个常数,以指出

下一条指令的地址，直到程序结束。指令顺序寻址的过程如图 3-11 所示。

（2）跳跃寻址方式。

当程序中出现分支或循环时，就会改变程序的执行顺序。此时，对指令寻址就要采取跳跃寻址方式。所谓跳跃，就是指下一条指令的地址不是通过程序计数器(PC)加 1 获得的，而是由指令本身给出的。指令系统中的无条件转移指令和各种条件转移指令，就是为跳跃寻址方式而设置的。跳跃寻址的过程如图 3-12 所示。

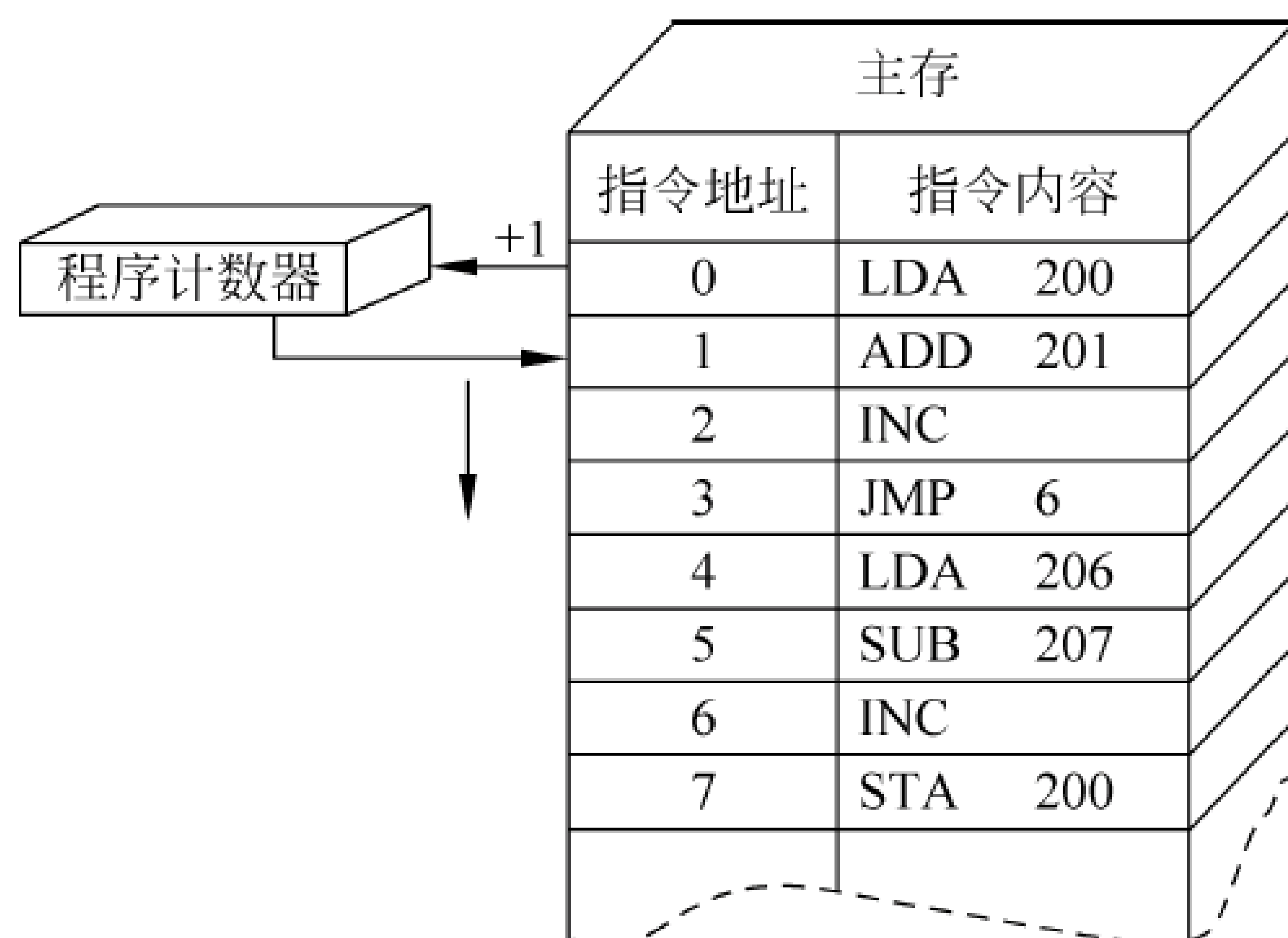


图 3-11　指令顺序寻址方式

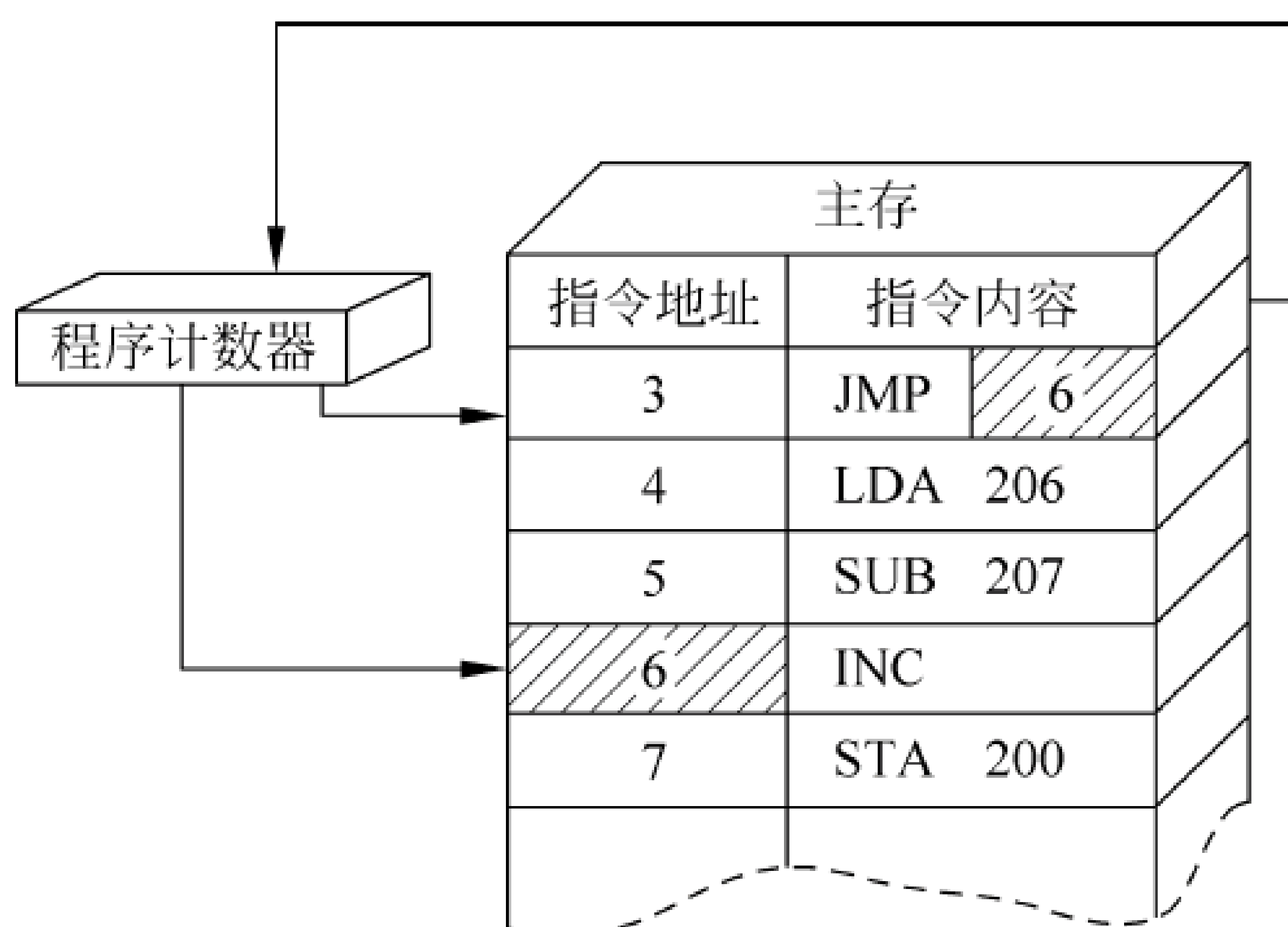


图 3-12　指令跳跃寻址方式

当执行 JMP 指令时，PC 不加 1，而是将 JMP 指令中的地址码 06 送到 PC，计算机从 06 地址取出指令执行，之后又顺序执行，直到再遇到转移指令或程序结束。

2. 操作数寻址方式

操作数寻址方式就是形成操作数有效地址的方法。由于指令长度的限制，指令中的地址码不会很长，而主存的容量却越来越大。因此把指令中地址码字段给出的地址称为形式地址，这个地址有可能不能直接用来访问主存，而经过某种运算后可得到能够直接访问主存的地址，称为有效地址(Effective Address，EA)。

设某计算机指令系统的单地址指令结构如图 3-13 所示。其中，OP 为操作码，X 为寻址特征码，D 为形式地址或称偏移量。寻址过程就是把 X 和 D 的不同组合变换成有效地址的过程。常用的寻址方式有：立即寻址、直接寻址、间接寻址、相对寻址、寄存器寻址、变址寻址等。

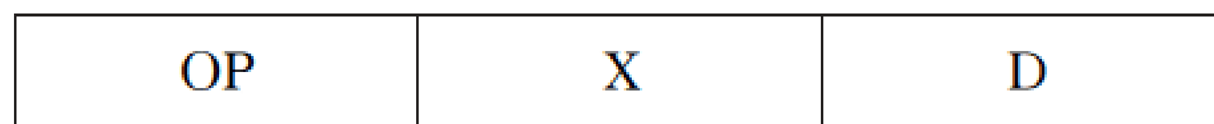


图 3-13　单地址指令结构

（1）立即寻址(Immediate Addressing)。

指令的地址码字段指出的不是操作数的地址，而是操作数本身，这种寻址方式称为立即寻址。这种寻址方式的特点是：在取指令时，操作码和操作数被同时取出，不必再次访问存储器，从而提高了指令的执行速度。但是，因为操作数是指令的一部分，不能被修改，而且立即数的大小受到指令长度的限制，所以这种寻址方式灵活性较差，通常用于给某一寄存器或主存单元赋初值。例如 Intel 80x86 的 MOV AX，2000H 指令，把立即数 2000H 传送给累

加寄存器 AX。

(2) 直接寻址(Direct Addressing)。

指令的地址码字段直接指出操作数的有效地址,这种寻址方式称为直接寻址。此时,特征码 X 指出寻址方式是直接寻址方式,形式地址 D 给出操作数的地址。若用 EA 代表有效地址,则有 EA=D。例如 Intel 80x86 的 MOV AX,[2000H]指令,源操作数的有效地址 EA=2000H。直接寻址过程如图 3-14 所示。

这种寻址方式不需要做任何寻址运算,简单直观,也便于硬件实现。但随着计算机主存容量的不断扩大,所需的地址码将会越来越长。对于定长指令,用于表示地址码字段位数有限,限制了访问主存的范围;而对于变长指令,势必造成指令的长度太长,因此这种寻址方式缺少灵活性。

(3) 间接寻址(Indirect Addressing)。

间接寻址是相对直接寻址而言的。间接寻址方式下,地址字段中的形式地址不是操作数的有效地址,而是操作数地址的地址。就是说,D 指示的存储单元中的内容才是操作数的有效地址,而 D 只是一个间接地址。此时,特征码 X 指示寻址方式为间接寻址方式,有效地址 EA=(D)。间接寻址过程如图 3-15 所示。

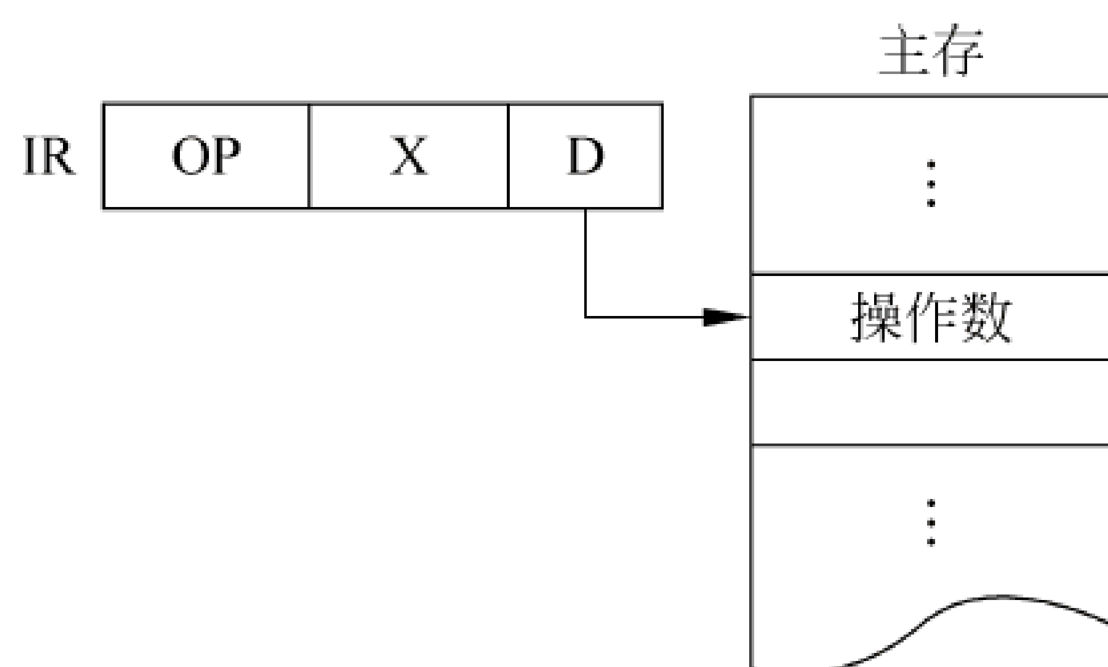


图 3-14 直接寻址方式

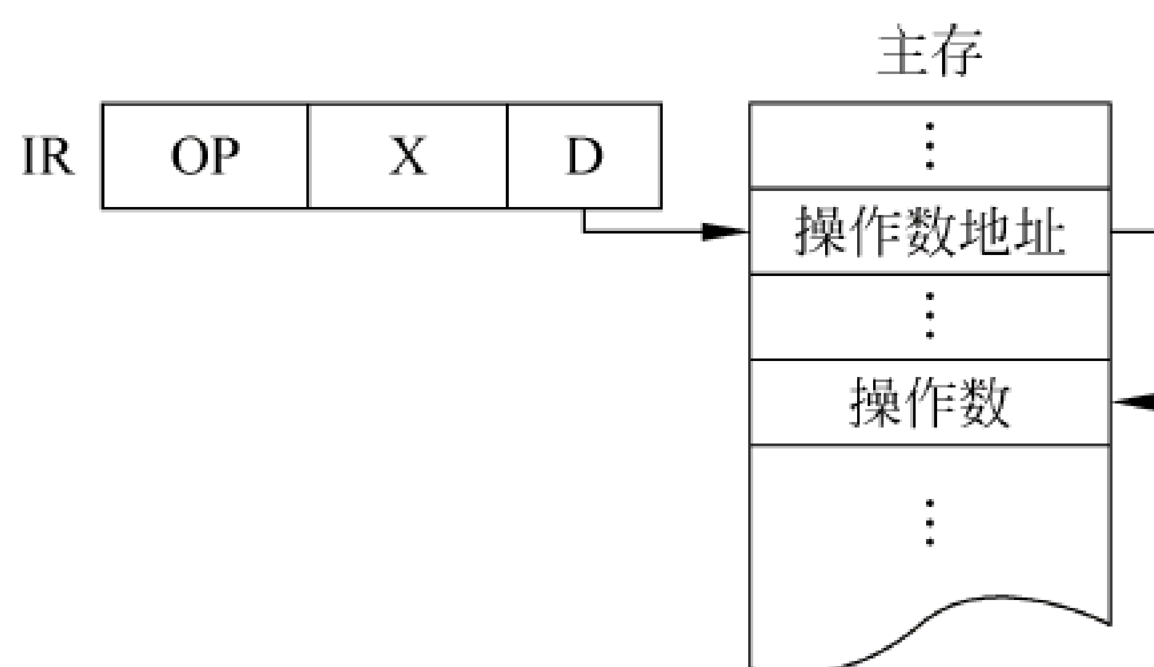


图 3-15 间接寻址方式

间接寻址扩大了寻址范围,可用较短的地址访问大的主存空间。例如:某计算机指令字长 16 位,指令中的地址码 10 位,直接寻址空间为 2^{10}(=1K)个存储单元。采用一级间接寻址后,从存储器中将取出 16 位的有效地址能够访问 2^{16}(=64K)个存储单元。

间接寻址中又有一级间址和多级间址之分,如图 3-15 所示的间接寻址是一级间址。多级间址为取得操作数需要多次访问主存,对于多级间址来说,在寻址过程中所访问到的每个主存单元的内容中都应设有一个间址标志位。通常将这个标志位放在主存单元的最高位。当该位为“1”,表示这一主存单元中仍然是间接地址,需要继续间接寻址;当该位为“0”时,表示已经找到了有效地址,根据这个地址可以找到真正的操作数。间接寻址在取指之后至少需要两次访问主存才能取出操作数,降低了指令执行的速度。所以,大多数计算机只允许一级间址。

(4) 寄存器寻址(Register Addressing)。

寄存器寻址是在指令的地址码字段给出某一个通用寄存器的编号,这个寄存器中存放着操作数。例如 Intel 80x86 的 MOV AX,DX 指令,源操作数存放在 DX 寄存器中,把它取出就可以传送给累加寄存器 AX。寄存器寻址过程如图 3-16 所示。

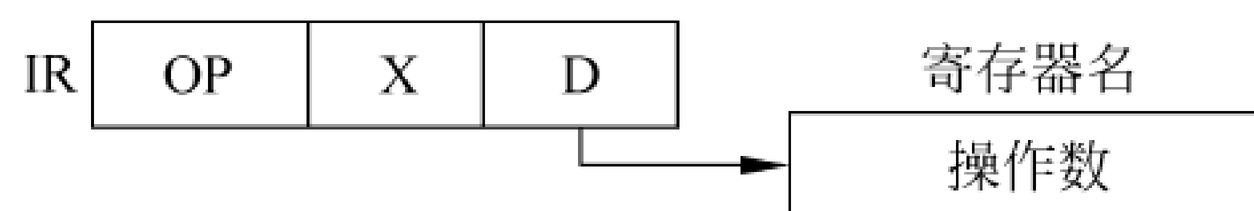


图 3-16　寄存器寻址方式

寄存器寻址的优点：从寄存器中存取数据比从主存中快得多；由于寄存器的数量较少，其地址码字段短，缩短了指令长度，提高了指令的执行速度，在各种计算机中广泛使用。

（5）寄存器间接寻址（Register Indirect Addressing）。

寄存器间接寻址是在指令的地址码中给出某一通用寄存器的编号，在寄存器中存放操作数的有效地址，而操作数则存放在主存单元中。X 特征码表示寻址方式为寄存器间接寻址方式，有效地址 EA＝（R）。例如 Intel 80x86 的 MOV AX，[BX]指令，源操作数的有效地址存放在 BX 寄存器中。寄存器间接寻址过程如图 3-17 所示。

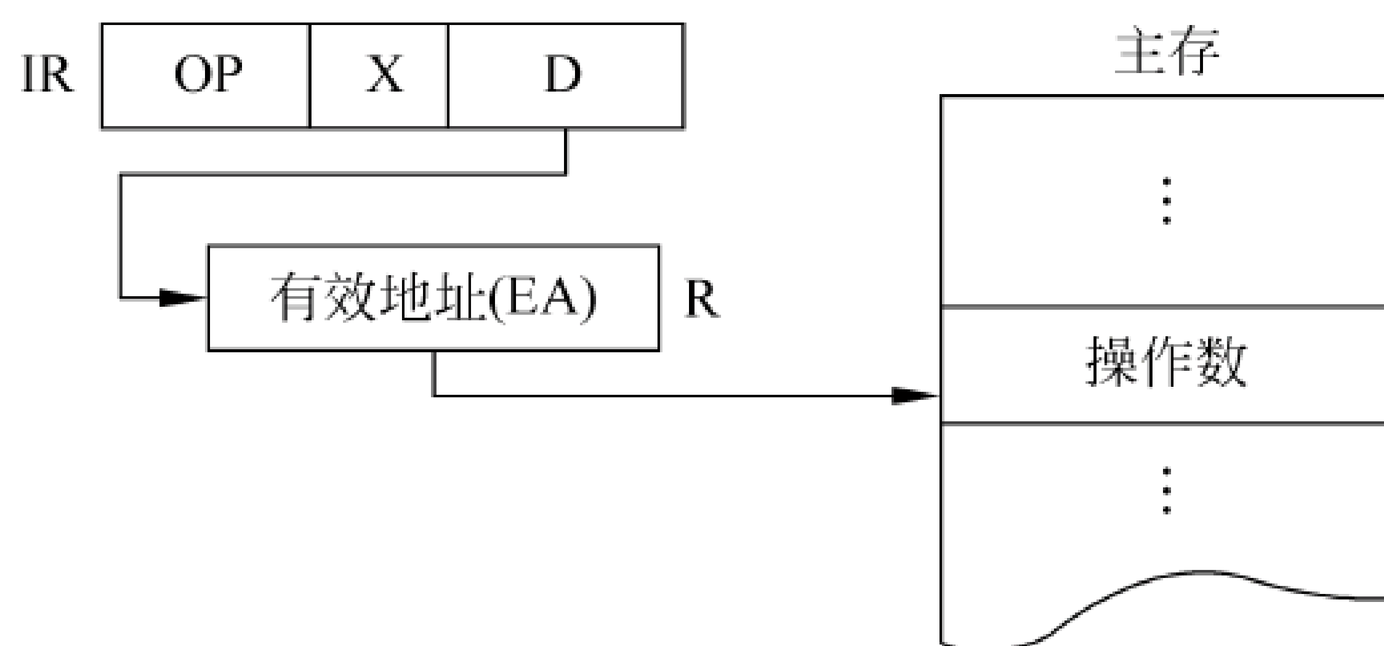


图 3-17　寄存器间接寻址方式

寄存器间接寻址方式的指令代码较短，克服了间接寻址中访存次数多的缺点，指令执行速度快，是一种广泛使用的寻址方式。此时寄存器本身被称为间接地址指示器，在编程过程中常作为地址指针。

（6）相对寻址（Displacement Addressing）。

相对寻址是把程序计数器（PC）中的内容加上指令中的形式地址 D，形成操作数的有效地址。此时，特征码 X 指示为相对寻址方式，PC 中的内容是当前的指令地址，这里的“相对”，就是相对当前的指令地址，所以，操作数的有效地址为 EA＝（PC）＋D。形式地址 D 可正、可负，通常用补码表示，操作数可在指令存储单元之前或之后。操作数的地址与指令地址之间总是相差一个固定值，支持程序在主存中任意浮动。寻址过程如图 3-18 所示。

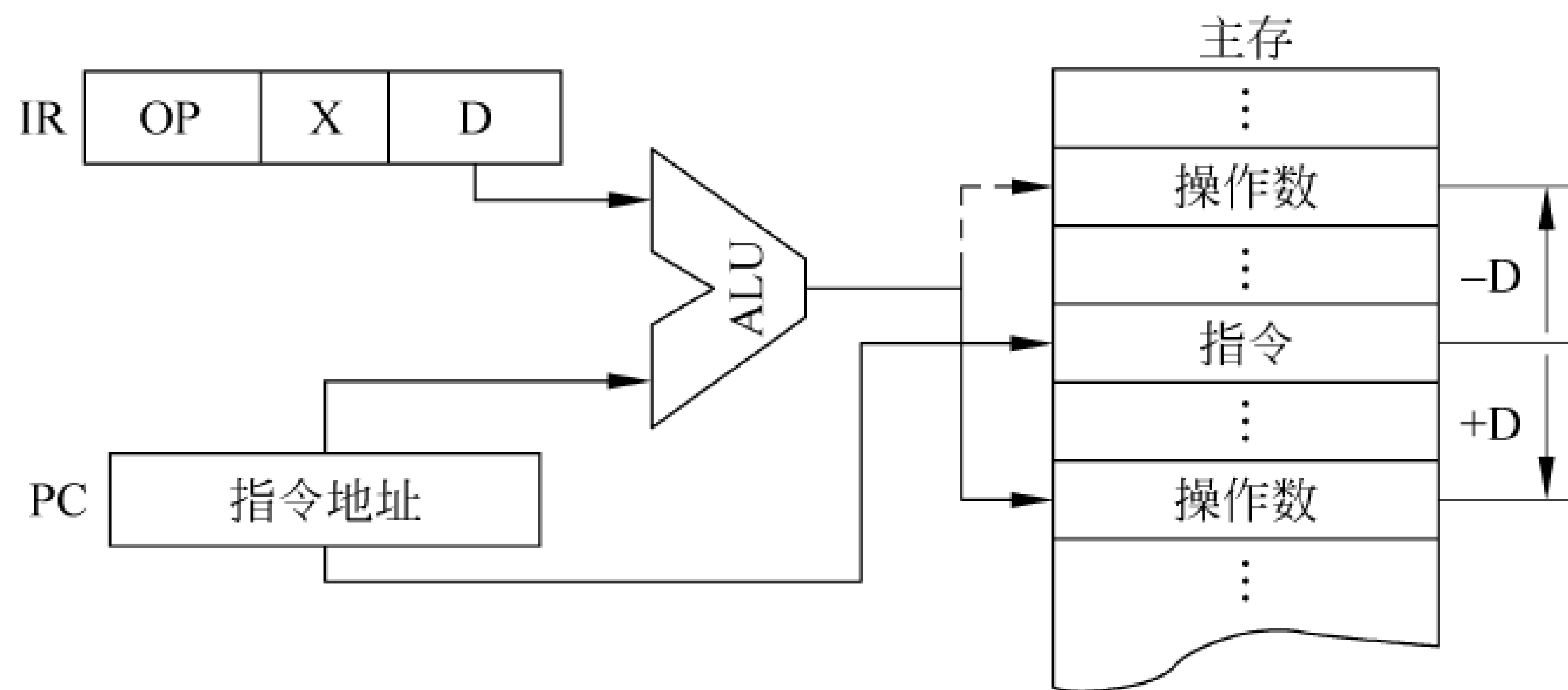


图 3-18　相对寻址方式

(7) 变址寻址(Index Addressing)。

使用一个变址寄存器的内容加上位移量 D 形成操作数的有效地址(EA),称为变址寻址。这种寻址方式与相对寻址的区别在于它与本条指令的地址无关,操作数地址随变址寄存器的内容浮动一个 D。变址寄存器的内容可由程序员设置,故寻址更灵活。在具有变址寻址的指令中,除去操作码和形式地址外,还必须指明变址寄存器。变址寻址的过程如图 3-19 所示。

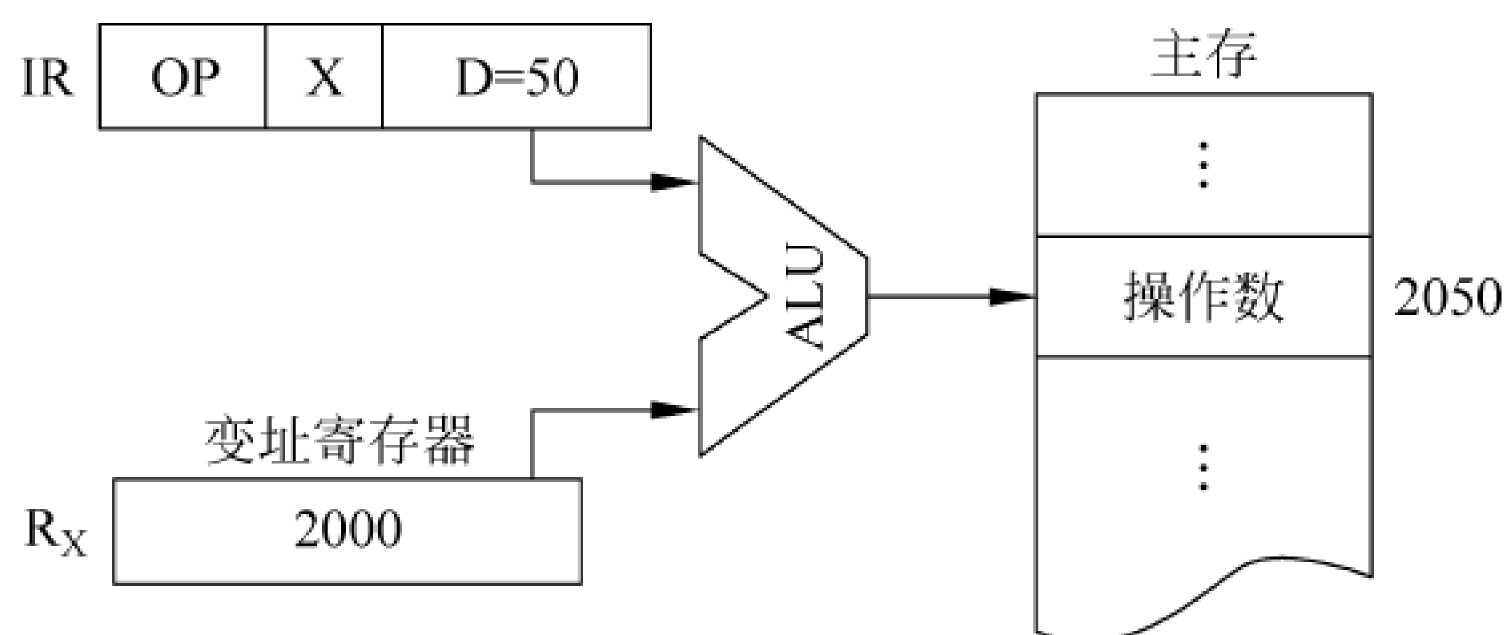


图 3-19 变址寻址方式

变址寻址是一种广泛采用的寻址方式,最典型的用法是将指令中的形式地址作为基准地址,而变址寄存器的内容作为修改量。在遇到需要频繁修改地址时无须修改指令,只要修改变址值就可以了,这对于数组运算、字符串操作等成批数据处理是很有用的。例如 Intel 80x86 的 MOV AX,TABLE[SI]指令,源操作数的有效地址 EA=(SI)+TABLE,SI 是变址寄存器,TABLE 是数组或字符串的基准地址,在此作为位移量 D。

(8) 基址寻址(Base Addressing)。

基址寻址是将基址寄存器 R_b 的内容与指令中给出的位移量 D 相加,形成操作数有效地址,即 EA=(R_b)+D。当特征码 X 指明是基址寻址时,要指定一个寄存器来存放寻址的基准值,这个寄存器称为基址寄存器,指令的地址码字段是一个可正、可负的位移量。基址寻址如图 3-20 所示。

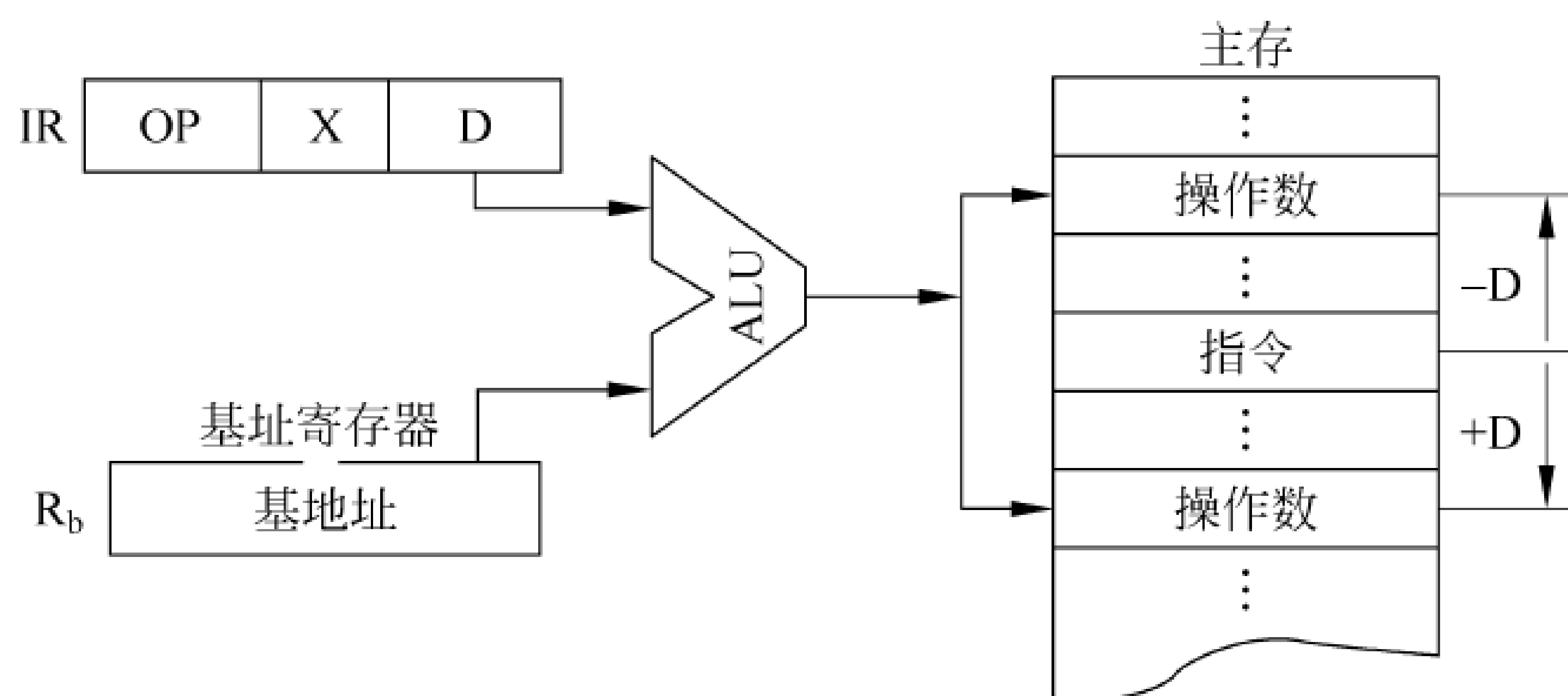


图 3-20 基址寻址方式

基址寻址和变址寻址在形成有效地址时所用的方法是相同的,而且在一些计算机中,这两种寻址方式都是用同样的硬件来实现的。它们两者的区别是:这两种寻址方式应用的场合不同,变址寻址是面向用户的,用于访问字符串、向量和数组等成批数据;而基址寻址面向系统,主要用于逻辑地址和物理地址间的变换,用以解决程序在主存中的再定位和扩大寻址空间等问题。

(9) 段寻址(Segment Addressing)。

段寻址是一种为了扩大寻址范围而采用的技术,在 Intel 8088/8086 等处理器中使用。8088/8086 处理器字长 16 位,因此,寻址范围只有 $2^{16}=64$KB,而处理器的直接寻址范围可达到 $2^{20}=1$MB,其中的奥妙就是采用了段寻址方式。具体方法是将 1MB 的存储空间以 64KB 为单位分为若干段,在形成操作数的有效地址时,由一个基地址加上某寄存器提供的 16 位偏移量以形成 20 位物理地址,这个基地址由 CPU 中的段寄存器提供。形成 20 位物理地址时,段寄存器中的内容自动左移 4 位,然后与偏移量相加,即形成所需的 20 位地址。段寻址过程如图 3-21 所示。

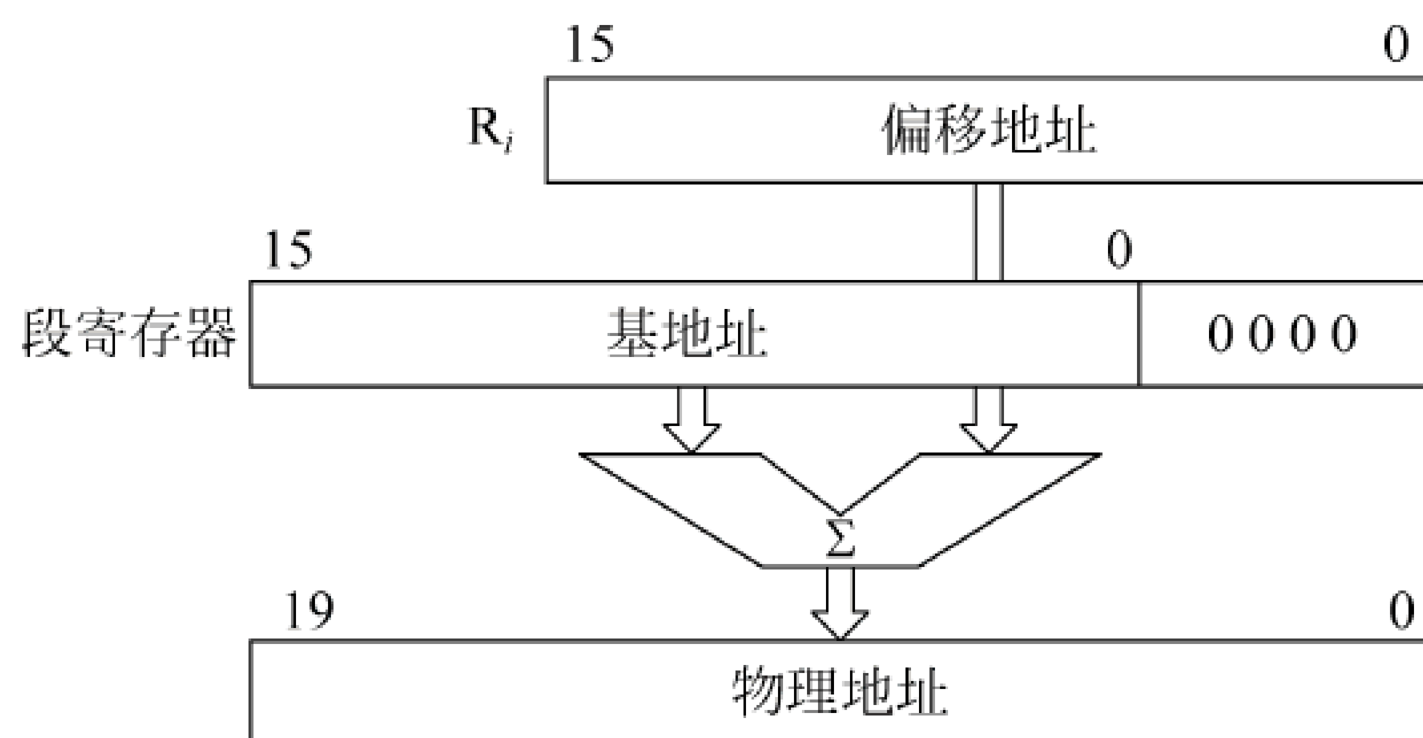


图 3-21　段寻址方式

(10) 复合寻址(Composite Addressing)。

复合寻址是把两种或两种以上的基本寻址方式组合在一起的寻址方式。如间接寻址方式与变址寻址或相对寻址方式结合而形成的寻址方式。

以上介绍了一些常用的基本寻址方式。对于某类型计算机的指令系统,可能只用了上述寻址方式的一部分或某些基本方式再加上几种变形的寻址方式。只要熟悉了以上的寻址方式,其他的寻址方式是很容易理解的。另外,在双地址指令或多地址指令中,各地址码的寻址方式可能不同,请读者注意。

【例 3.4】　一种二地址 RS 型指令的结构如图 3-22 所示。

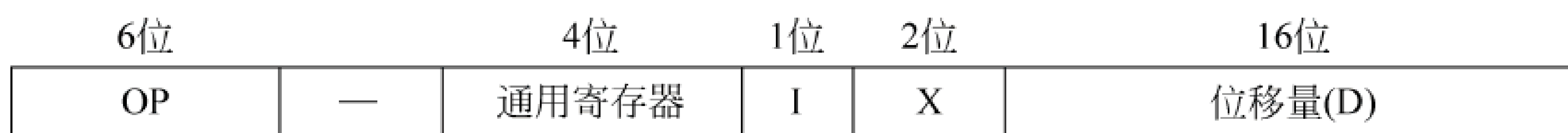


图 3-22　RS 型指令结构

其中 I 为间接寻址标志位,X 为寻址模式字段,D 位移量字段。通过 I,X,D 的组合,可构成如表 3-3 所示的寻址方式。请写出表中 6 种寻址方式的名称。

表 3-3　寻址方式

寻址方式	I	X	有效地址 EA 算法	说　明
(1)	0	00	EA=D	
(2)	0	01	EA=(PC)+/−D	PC 为程序计数器
(3)	0	10	$EA=(R_2)+/-D$	R_2 为变址寄存器
(4)	1	11	$EA=(R_3)$	
(5)	1	00	EA=(D)	
(6)	0	11	$EA=(R_1)+/-D$	R_1 为基址寄存器

解：(1) 直接寻址；　　(2) 相对寻址；　　(3) 变址寻址；
(4) 寄存器间接寻址；　　(5) 间接寻址；　　(6) 基址寻址。

3. 堆栈寻址方式

最后考虑堆栈寻址(Stack Addressing)方式。堆栈(Stack)是一种位置的线性序列，它的存取顺序是"后进先出"(Last In First Out)。堆栈是一个预留的位置块，数据项是被陆续加到栈顶的，在任何时刻，堆栈的位置块是部分被填充的。堆栈寻址方式是一种隐含寻址，隐含地指示对栈顶位置的数据执行操作。

1) 堆栈的设置

堆栈的设置通常有两种方式：硬堆栈和软堆栈。硬堆栈是利用 CPU 中的一组专门的寄存器来组成的，硬堆栈的容量有限。软堆栈是通过执行相关指令，把主存储器中的一段存储区定义为堆栈，利用一个通用寄存器作为堆栈指针(Stack Pointer，SP)，并设置 SP 的初值。目前大多数计算机中都支持软堆栈，软堆栈设置灵活，可满足较大容量的要求，可以用对存储器寻址的指令来对堆栈中的数据进行访问。

2) 堆栈的工作方式

将数据存入堆栈的操作称为"进栈(PUSH)"，从堆栈中取出数据的操作称为"出栈(POP)"，出栈后相应单元的数据不再保留。最先进栈的数据位于堆栈的底端(栈底)，最后进栈的数据位于堆栈的顶端(栈顶)，进栈和出栈操作均在栈顶进行。图 3-23 表示了"进栈"操作的情况。

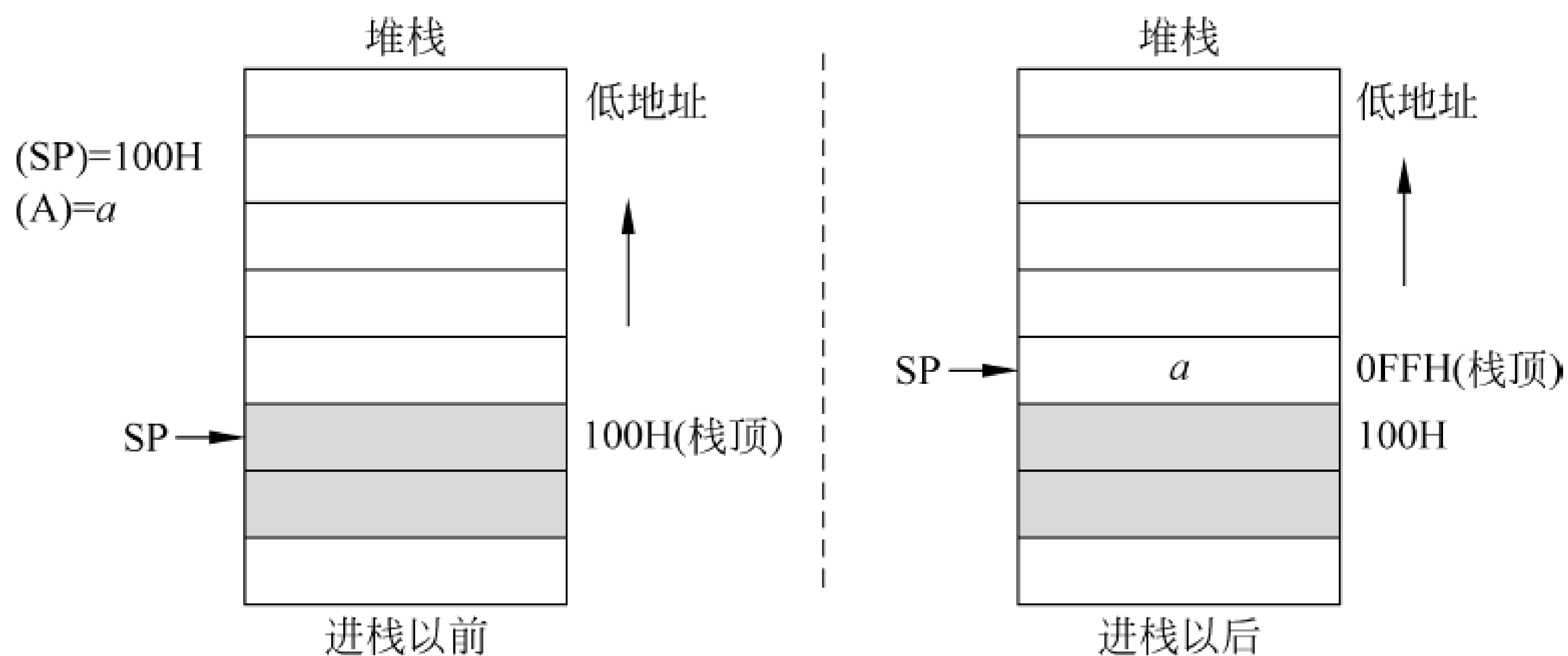


图 3-23　进栈操作

在执行进栈操作前，设堆栈指针(SP)＝100H，通用寄存器(A)＝a，假如现在把通用寄存器 A 中的数据送入堆栈，那么执行"进栈"指令后，SP 自动"减 1"成为 0FFH(十六进制)，数据 a 被压入存储单元 0FFH，SP 始终指向栈顶(最后存入数据的堆栈单元)，因此进栈操作可描述如下：

$$SP \leftarrow (SP)-1, M_{SP} \leftarrow (A)$$

其中(A)表示通用寄存器 A 的内容，SP 表示堆栈指针，M_{SP}表示堆栈指针指示的栈顶单元。

图 3-24 表示了"出栈"操作的情况。

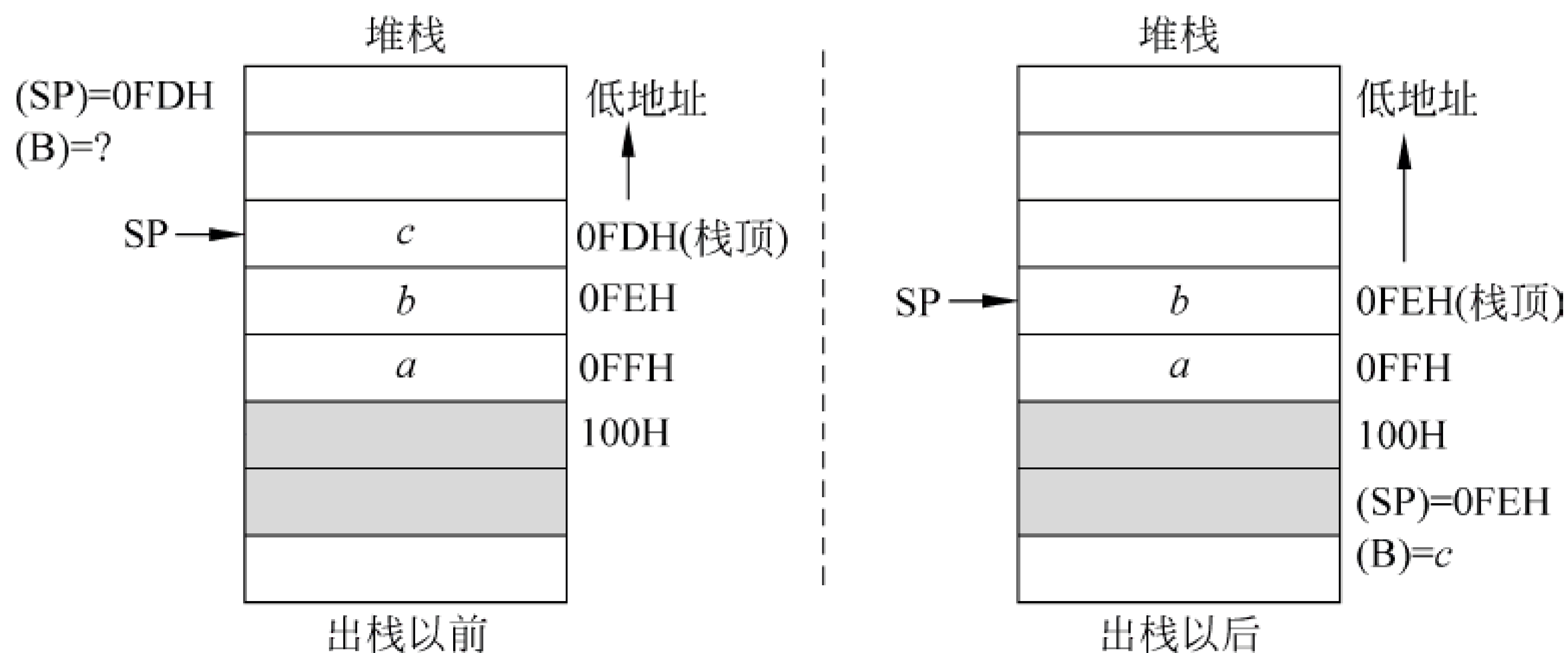


图 3-24　出栈操作

假定出栈操作之前，堆栈中已存入 a、b、c 三个数，堆栈指针(SP)＝0FDH，要求把堆栈中的数据送到通用寄存器 B，那么执行“出栈”操作后，就把当前栈顶单元的内容 c 送到了 B，堆栈指针 SP 修改为 0FEH。

出栈操作可描述如下：

$$B \leftarrow (M_{SP}), SP \leftarrow (SP)+1$$

要注意的是，有的机器进栈操作时，SP 由低地址向高地址变化；出栈操作时，SP 由高地址向低地址变化。

除了利用堆栈暂时存取数据外，在执行转子程序指令、中断指令要用到堆栈，子程序的嵌套、递归调用也离不开堆栈。

3.4　RISC

从计算机指令系统设计的角度看，计算机的体系结构可分为复杂指令系统计算机(Complex Instruction Set Computer，CISC)和精简指令系统计算机(Reduced Instruction Set Computer，RISC)。CISC 仍是当前计算机体系结构的主流，而 RISC 则是近 20 年来迅速发展的一种体系结构。本节介绍 RISC 的发展和特点。

长期以来，计算机性能的提高往往是通过增加硬件的复杂性来获得的。随着集成电路技术，特别是超大规模集成电路(VLSI)技术的迅速发展，为了软件编程方便和提高程序的运行速度，硬件工程师采用的办法是不断增加可实现复杂功能的指令和多种灵活的寻址方式，甚至某些指令可支持高级语言语句归类后的复杂操作，致使硬件越来越复杂，造价也相应提高。为实现复杂操作，处理器除向程序员提供各种寄存器和复杂功能机器指令外，还通过存于只读存储器(ROM)中的微程序来实现其极强的功能，处理器在分析每一条指令之后执行一系列微指令来完成所需的功能，这种设计称为复杂指令集计算机结构。一般 CISC 计算机所含的指令数目至少有 300 条以上，有的甚至超过 500 条。

3.4.1　精简指令集计算机的出现

采用复杂指令系统的计算机有着较强的处理高级语言的能力，这对提高计算机的性能是有益的。当计算机的设计沿着这条道路发展时，有些人回过头去看一看过去走过的道路，

开始怀疑这种传统的做法：IBM 公司设在纽约的研究中心于 1975 年组织力量研究指令系统的合理性问题，因为当时已感到，日趋庞杂的指令系统不但不易实现，而且还可能降低系统性能。1979 年以 Patterson 教授为首的一批科学家也开始在美国加州大学伯克莱分校开展这一研究，结果表明，CISC 存在许多缺点。

首先，在这种计算机中，各种指令的使用频率相差悬殊：一个典型程序的执行过程所使用的 80%指令，只占一个处理器指令系统的 20%。事实上最频繁使用的指令是存数、取数和加运算这些最简单的指令，这样一来，长期致力于复杂指令系统的设计，实际上是在设计一种难得在实践中用得上的指令系统的处理器。同时，复杂的指令系统必然带来结构的复杂性，这不但增加了设计的时间与成本，还容易造成设计失误。

此外，在 CISC 中，许多复杂指令需要极复杂的操作，这类指令多数是某种高级语言的直接翻版，因而通用性差，由于采用二级的微程序执行方式，它也降低了那些被频繁使用的简单指令的运行速度。

因此，针对 CISC 的这些弊病，1975 年 IBM 公司 John Cocke 提出了精简指令系统的设想。到了 1979 年，美国加州大学伯克莱分校由 Patterson 教授领导的研究小组，首先提出了 RISC 这一术语，即指令系统应当只包含那些使用频率很高的少量指令，并提供一些必要的指令以支持操作系统和高级语言，按照这个原则发展而成的计算机被称为精简指令集计算机，简称 RISC。Patterson 教授领导的研究小组先后研制了 RISC-Ⅰ 和 RISC-Ⅱ 计算机。1981 年美国的斯坦福大学在 Hennessy 教授领导下的研究小组研制了 MIPS RISC，强调高效的流水线和采用编译方法进行流水调度，使得 RISC 技术设计风格得到很大补充和发展。

3.4.2 精简指令集计算机特点

RISC 是在 CISC 的成功技术并克服 CISC 缺点的基础上产生并发展起来的。一般认为，RISC 计算机具有下列特点。

1. 充分利用 VLSI 芯片的面积

CISC 机器的控制器大多采用微程序控制，其控制存储器在 CPU 芯片内所占的面积为 50%以上，而 RISC 机器的控制器采用组合逻辑控制，其硬布线逻辑只占 CPU 芯片面积的 10%左右，可见它可将空出的面积供其他功能部件使用。这样，在相同的集成电路制造技术条件下，同样的芯片面积上可实现更多的并行部件，如超级流水线、多核处理器。

2. 提高计算机运算速度

RISC 机器提高运算速度，主要反映在以下 5 个方面：

(1) RISC 指令系统采用固定长度的指令格式，选取使用频率最高的一些简单指令，指令种类少，指令规整、简单、基本寻址方式有 2～3 种。

(2) RISC 机器内大量使用寄存器，数据处理指令只对寄存器进行操作，只有加载/存储指令可以访问存储器，以提高指令的执行效率。

(3) RISC 机器采用寄存器窗口重叠技术，程序嵌套时不必将寄存器内容保存到存储器中，又提高了执行速度。

(4) RISC 机器采用组合逻辑控制比采用微程序控制的 CISC 机器的延迟小，缩短了

CPU 周期。

（5）RISC 机器选用精简指令系统，适合于流水线工作，大多数指令在一个时钟周期内完成。

3. 便于设计，可降低成本，提高可靠性

RISC 指令系统简单，故机器设计周期短，逻辑简单，设计出错可能性小，有错也容易发现，可靠性高。

4. 有效支持高级语言程序

由于 RISC 指令系统指令少，寻址方式少，使编译程序容易选择更有效的指令和寻址方式。而且由于 RISC 机器的通用寄存器多，可尽量安排寄存器的操作，使编译程序的代码优化效率提高。RISC 指令系统靠优化编译来更有效地支持高级语言程序。

目前在中高档服务器中普遍采用这种指令系统的处理器，特别是高档服务器全都采用 RISC 指令系统的 CPU，RISC 指令系统更加适合高档服务器的 UNIX 和类 UNIX 操作系统。在中高档服务器中采用 RISC 指令的 CPU 主要有：PowerPC 处理器、SPARC 处理器、PA-RISC 处理器、MIPS 处理器、Alpha 处理器等。

当然，和 CISC 结构相比较，尽管 RISC 结构有上述的优点，但绝不能认为 RISC 结构就可以完全取代 CISC 结构，事实上，RISC 和 CISC 各有优势，而且界限并不那么明显。现在有些典型的 CISC 处理器中，内部加入 RISC 的特性，如 Intel 公司的 80486、Pentium 系列 CPU 就融合了 RISC 和 CISC 的优势。

3.5　指令系统举例

前面几节主要介绍了指令的格式、寻址方式、指令的操作类型，本节通过对 Pentium 指令系统和 SPARC 指令系统的简介，加深读者对指令系统的理解。

3.5.1　Pentium 微处理器指令系统

Pentium 指令集包含 8086/8088 以及 80486 的全部指令，其目的是为了实现兼容性。但 Pentium 指令集在其基础上增加了一些指令，有些指令的操作数可以是 32 位，有些指令的功能得到了扩充。下面对其指令结构作详细的分析介绍。

1. Pentium 的数据类型

操作数是指令的操作对象，在 Pentium 中，可处理的数据类型的长度有字节、字、双字、四字，还有一些特殊的数据类型，如表 3-4 所示。

2. Pentium 指令格式

Pentium 指令集所有指令的长度一般在 1～12 个字节间变化，它根据需要包括一个或多个前缀（限定指令操作时的属性），这种非固定长度的指令格式是典型的 CISC 结构特征。Pentium 指令集的变长指令格式，一是为了与它的前身 80486 保持兼容，二是希望能给编译

器的编写者提供更有效的代码，Pentium 指令格式如图 3-25 所示。

表 3-4 Pentium 的数据类型

数据类型	说　明
无符号数	包括 8 位/16 位/32 位无符号数，分别对应于字节、字、双字
有符号数	以补码形式表示，其最高位是符号位，其余位是数值位，支持 8 位、16 位、32 位、64 位（浮点单元）有符号整数
浮点数	有单精度实数、双精度实数、扩展精度实数
BCD 数	组合的 BCD 数据（每个字节包含两位十进制数），非组合的 BCD 数据（每个字节只含一位十进制数，高四位为 0）
串数据	串数据是一串连续的数据，可由字节组成也可由字或双字组成
ASCII 码	ASCII 码是基于拉丁字母的一套电脑编码系统，主要用于显示现代英语和其他西欧语言，它是现今最通用的单字节编码系统，并等同于国际标准 ISO/IEC 646
指针数据	指针数据指出给定数据在存储器中的地址，包括：近程指针——32 位逻辑地址，即 32 位段内偏移量，用于段内数据访问或段内转移；远程指针——48 位逻辑地址，即由 16 位段选择符和 32 位偏移量组成，用于段间数据访问或段间转移

0或1	0或1	0或1	0或1(字节数)
指令前缀	段取代	操作数长度取代	地址长度取代

(a) 前缀

1或2	0或1						0或1	0/1/2/4	0/1/2/4(字节数)
操作码	Mod	Reg或操作码	R/M	比例S	变址I	基址B	位移量	立即数
	2位	3位	3位	2位	3位	3位		

(b) 指令

图 3-25 Pentium 指令格式

下面分别具体介绍。

前缀是可选项，其作用是对其后的指令本身进行约定。这些前缀并不改变执行的内容，但使用这些前缀之后，指令就有了新的属性。

指令前缀——包括 LOCK（锁定）前缀和重复前缀。LOCK 前缀用于多处理器环境中对共享存储器的排他性访问。重复前缀用于字符串的重复操作，以获得比软件循环方法更快的速度。

段取代前缀——根据指令的定义和程序的上下文，一条指令所使用的段寄存器名称可以不出现在指令格式中，这称为段缺省规则。当要求一条指令不按缺省规则使用某个段寄存器时，必须用段取代前缀明确指明此段寄存器。

操作数长度取代前缀和地址长度取代前缀——在实地址模式下，操作数和地址的默认长度是 16 位；在保护模式下，由段描述符中的 D 位来确定默认长度，若 $D=1$，操作数和地址的默认长度是 32 位，若 $D=0$，二者的默认长度是 16 位。

指令本身由操作码字段、Mod-R/M 字段、SIB 字段、位移量字段、立即数字段组成。除操作码字段外，其他 4 个字段都是可选字段。

主操作码——主操作码字段占一到两个字节，这个操作码可以定义较小的编码字段。

Mod-R/M——规定了存储器操作数的寻址方式，给出了寄存器操作数的寄存器地址

号。Pentium 指令集中几乎所有的引用存储器操作数的指令都有这个字段，它占用寻址方式字节的高两位(Mod)和低三位(R/M)，用来显示指定指令所需操作数的寄存器或是存储器寻址方式，比如默认的段选择符、是否有偏移量等，计算有效地址的偏移量、基址、变址、比例因子等信息由寻址方式字段指定。

SIB——该域占用一个字节，只作为 32 位寻址方式(基地址＋比例×变址＋Disp)的补充。SIB 字节的第 7 和第 6 位是比例域。在所有的情况下，比例域可能的取值为 00、01、10、11 这 4 种情况，对应的比例值分别为 1、2、4、8。SIB 字节的第 5～第 3 位是变址域，它指定一个 32 位的寄存器为变址寄存器。SIB 字节的第 2～第 0 位是基址域，它指出用作有效地址的基地址寄存器操作数。

位移量——指令中直接给出寻址方式所需的位移量。当为存储器寻址时，它是一个计算有效地址或程序相对地址的常量，根据 W 位以及当前操作模式 16 位/32 位，位移量可以是 1、2、4 字节。一般位移量有三种形式：有符号相对位移量，无符号相对位移量和绝对位移量。

立即数——指令中直接给出操作数，其范围是从零字节到四个字节。如果在编码中有这一个域，则包含一个被用作指令操作数或自变量的常量。

3. Pentium 的寻址方式

Pentium 的外部地址总线宽度是 36 位，但它也支持 32 位物理地址空间。在实地址模式下，逻辑地址形式为段寻址方式：将段名所指定的段寄存器内容(16 位)左移 4 位，低 4 位补全 0，得到 20 位段基地址，再加上 16 位段内偏移，即得 20 位物理地址。在保护模式下，32 位段基地址加上段内偏移得到 32 位线性地址，由存储管理部件将其转换成 32 位的物理地址。

Pentium 主要有 9 种寻址方式，其中有 7 种实现对存储器操作数寻址。Pentium 的寻址方式如表 3-5 所示。

表 3-5 Pentium 的寻址方式

序号	寻址方式名称	有效地址 EA 计算	说明
(1)	立即寻址		操作数在指令中
(2)	寄存器寻址		操作数在某寄存器中，指令给出寄存器号
(3)	直接寻址	EA＝Disp	Disp 为偏移量
(4)	基址寻址	EA＝(B)	B 为基址寄存器
(5)	基址＋偏移量	EA＝(B)＋Disp	
(6)	比例变址＋偏移量	EA＝(I) * S＋Disp	I 为变址寄存器，S 为比例因子
(7)	基址＋变址＋偏移	EA＝(B)＋(I)＋Disp	
(8)	基址＋比例变址＋偏移量	EA＝(B)＋(I) * S＋Disp	
(9)	相对寻址	指令地址＝(PC)＋Disp	PC 为程序计数器或当前指令指针寄存器

下面对 32 位寻址方式作几点说明。

(1) 立即数可以是 8 位、16 位、32 位。

(2) 寄存器地址：一般指令或使用 8 位通用寄存器，或使用 16 位通用寄存器，或使用 32 位通用寄存器。对 64 位浮点数操作，要使用一对 32 位寄存器。少数指令以段寄存器来实施寄存器寻址方式。

(3) 直接寻址：也称偏移量寻址方式，偏移量长度可以是 8 位、16 位、32 位。

(4) 基址寻址：基址寄存器 B 可以是上述通用寄存器中的任何一个。基址寄存器 B 的内容为有效地址。

(5) 基址＋偏移量寻址：基址寄存器 B 是 32 位通用寄存器中的任何一个。

(6) 比例地址＋偏移量寻址：也称为变址寻址方式，变址寄存器 I 是 32 位通用寄存器中除 ESP 外的任何一个，而且可将此变址寄存器内容乘以 1、2、4 或 8 的比例因子 S，然后再加上偏移量而得到有效地址。

(7)、(8)两种寻址方式是(4)、(6)两种寻址方式的组合，此时偏移量可有可无。

(9) 相对寻址：适用于转移控制类指令。用当前指令指针寄存器 EIP 或 IP 的内容(下一条指令地址)加上一个有符号的偏移量，形成 CS 段的段内偏移。

4. Pentium 操作类型

Pentium 指令集完全包含 80486 指令集，而 80486 指令系统是一种典型的 CISC 指令，全部指令可分为三大类：整型类指令、浮点类指令、特权指令。

整型类指令完成对整数的操作，具体又可以分为若干组：数据传送、二/十进制算术运算、逻辑操作、循环移位操作、字符串操作、位测试与字节操作、控制转移操作、标志控制、段寄存器以及其他指令。

浮点类指令是由浮点部件(FPU)所执行的指令，用来对浮点数、扩展的整形数、二进制编码的十进制(BCD)数等操作数进行操作。

特权指令用于实现系统级的功能，像加载系统级寄存器(GDTR、IDTR、LDTR 等)、管理高速缓存、管理中断或设置调试寄存器等。Pentium 的操作类型如表 3-6 所示。

表 3-6　Pentium 的操作类型

类　型	指　令	操作说明
数据传送	MOV	在寄存器和存储器之间或寄存器之间传送数据
	PUSH/POP	将数据进栈/出栈
	PUSHA	将所有通用寄存器的内容压入堆栈
	MOVSX	传送字节、字、双字，符号被扩展
	LEA	装入有效地址
	XLAT	代码转换
	IN/OUT	输入输出
算术运算	ADD	加法
	SUB	减法
	MUL	乘法
	IDIV	除法
逻辑运算	AND	逻辑与
	BTS	位测试并置位
	BSF	扫描一个字节或字
	SHL/SHR	逻辑左移/逻辑右移
	SAL/SAR	算术左移/算术右移
	ROL/ROR	循环左移/循环右移
	SET_{CC}	根据状态标志定义的 16 条件中的一个

续表

类 型	指 令	操 作 说 明
控制转移	JMP CALL JE/JZ LOOPE/LOOPZ INT/INTO	无条件转移 转子程序，并把断点压入堆栈 如果相等则转移 如果相等则循环 中断指令
串操作	MOVS LODS	传送字节、字、双字的串 加载字节、字、双字的串
高级语言	ENTER	生成一个可以用来实现块结构高级语言规则的栈
支持	LEAVE BOUND	为 ENTER 的逆 检查阵列的边界
标志控制	STC LAHF	置进位标志 将标志加载到 AH 寄存器
段寄存器	LDS/ LES/ LSSLFS/ LGS HLT LOCK	将指针送寄存器和 DS、ES、SS、FS、GS 保持原状态 封锁总线，Pentium 可独占存储器
系统控制	ESC WAIT	指示后面的指令支持高精度整数和浮点数计算 处理器处于等待状态，直到 BUSY 信号撤除
保护	SGDT LSI VERR/VERW INVD WBINVD INVLPG	保存全局描述符表 将段限装入一个用户指定的寄存器中 读/写核查段 将内部 cache 清空 将更改行写回寄存器后将内部 cache 清空 使一个转换后的 TLB 无效

3.5.2 SPARC 指令系统

最早采用 RISC 思想的计算机系统是 IBM 801，以后又研制成 RISC-Ⅰ、RISC-Ⅱ及 MIPS 计算机等。目前的许多处理机都采用了 RISC 体系结构，如 Sun 公司的 SPARC、Super SPARC、Ultra SPARC。SGI 公司的 R4000、R5000、R10000，IBM 公司的 Power、Power PC，DEC 公司的 Alpha，HP 公司的 HP3000/930 系列、950 系列等。下面以 SPARC 为例来说明 RISC，它是一个 32 位字长的计算机，共有 75 条指令。

1. 指令类型和指令格式

SPARC 机有以下 6 种指令类型：

(1) 算术运算、逻辑运算、移位指令；

(2) 取数(LOAD)/存数(STORE)指令；

(3) 控制转移指令；

(4) 读/写专用寄存器指令；

(5) 浮点运算指令；

(6) 协处理器指令。

SPARC 共有 3 种指令格式，如表 3-7 所示。

表 3-7 SPARC 的 3 种指令类型

<table>
<tr><td rowspan="2">格式 1
CALL 指令</td><td colspan="7">31 30 29 28 0</td></tr>
<tr><td>OP</td><td colspan="6">Disp30(偏移量)</td></tr>
<tr><td rowspan="3">格式 2
BRANCH
SETHI 指令</td><td colspan="7">31 30 29 28 25 24 22 21 0</td></tr>
<tr><td>OP</td><td>a</td><td>Cond</td><td>OP_2</td><td colspan="3">Disp30(偏移量)</td></tr>
<tr><td>OP</td><td colspan="2">R_d</td><td>OP_2</td><td colspan="3">imm22(立即数)</td></tr>
<tr><td rowspan="4">格式 3
其他指令</td><td colspan="7">31 30 28 25 24 19 18 14 13 12 5 4 0</td></tr>
<tr><td>OP</td><td>R_d</td><td>OP_3</td><td>R_{S1}</td><td>i</td><td>A_{S1}</td><td>R_{S2}</td></tr>
<tr><td>OP</td><td>R_d</td><td>OP_3</td><td>R_{S1}</td><td>i</td><td colspan="2">Simm13</td></tr>
<tr><td>OP</td><td>R_d</td><td>OP_3</td><td>R_{S1}</td><td colspan="2">OP_f</td><td>R_{S2}</td></tr>
</table>

其中 OP、OP_2、OP_3 为指令操作码，OP_f 为浮点指令操作码。为了增加立即数和偏移量的长度，有 3 条指令的操作码缩短了，其中 CALL 是子程序调用指令，BRANCH 是转移类指令，SETHI 指令的功能是将 22 位立即数左移 10 位，送入 R_d 所指示的寄存器中，然后再执行一条加法指令以补充后面 10 位二进制数据，从而形成 32 位字长的数据。

R_{S1}、R_{S2} 为通用寄存器地址，用作源操作数寄存器地址或地址寄存器地址。

R_d 为目标寄存器地址，目标寄存器用来保存运算结果或用来存储器取来的数据。但是在执行存数指令时，保存的则是源操作数，并将此操作数送往指定的存储器单元地址中。

Simm13 是 13 位扩展符号的立即数。运算时若其最高位为 1，则最高位前面的所有位都扩展为 1；若其最高位为 0，则最高位前面的所有位都扩展为 0。

i 用来选择第二个操作数。$i=0$ 时，第二操作数在 R_{S2} 中；$i=1$ 时，第二操作数为 Simm13。

2. 指令的功能与寻址方式

1）算术逻辑运算指令

功能：将 R_{S1}、R_{S2} 的内容（或 Simm13）按操作码规定的操作运算后将结果送往 R_d。

当 $i=0$ 时，$R_d \leftarrow (R_{S1})OP(R_{S2})$；当 $i=1$ 时，$R_d \leftarrow (R_{S1})OP$ Simm13。

2）取数/存数指令

功能：LOAD 指令将存储器中的数据送往 R_d 中，而 STORE 指令将 R_d 中的数据送往存储器中。

存储器地址的计算方法如下（寄存器间接寻址方式）：

当 $i=0$ 时，存储器地址 $=(R_{S1})+(R_{S2})$；

当 $i=1$ 时，存储器地址 $=(R_{S1})+$Simm13。

3）控制转移类指令

条件转移（BRANCH）：由 Cond 字段决定程序是否转移，用相对寻址方式形成转移地址。

转移并连接（JMPL）：将本条指令的地址（PC 值）保存在以 R_d 为地址的寄存器中以备程序返回时使用。用寄存器间址方式形成转移地址。

调用（CALL）：采用相对寻址方式形成转移地址。

陷阱（TRAP）：采用寄存器间址方式形成转移地址。

从 TRAP 程序返回(RETT)：用寄存器间址方式形成转移地址。

4）读/写专用寄存器指令

SPARC 有 4 个专用寄存器(PSR，Y，WIM，TBR)，其中 PSR 称为程序状态寄存器，它的内容反映并控制机器的运行状态，比较重要，因此读/写 PSR 的指令一般是特权指令。

在 SPARC 中，有一些基本操作没有选入指令系统，但很容易使用指令集中已有的一条指令来替代实现。表 3-8 左半部列出了 6 种基本操作的功能，表的右半部是替代指令及实现方法。因为 SPARC 约定 R_0 的内容恒为 0，而且立即数可以作为一个操作数处理，所以某些操作可以替代实现，由此可体验到所谓“精简指令系统”的含义和用意。

表 3-8　SPARC 中的基本操作替代实现

基本操作	替换指令	实现方法
寄存器间传送数据	ADD	$R_s+R_0 \rightarrow R_d$
寄存器内容加 1	ADD	立即数 imm13＝1，作为操作数
寄存器内容减 1	SUB	立即数 imm13＝－1，作为操作数
取负数	SUB	$R_d \leftarrow R_0-R_s$
取反码	XOR	立即数 imm13＝－1，作为操作数
清除寄存器	ADD	$R_d \leftarrow R_0+R_0$

本章小结

本章主要讲述了以下内容：

(1) 指令系统是一台计算机中所有机器指令的集合，它是表征一台计算机性能的重要因素，指令的格式与功能不仅直接影响到机器的硬件结构，而且也影响到系统软件。

(2) 指令格式是指令用二进制代码表示的结构形式，通常由操作码字段和地址码字段组成。操作码字段表征指令的操作特性与功能，而地址码字段指示操作数的地址。目前多采用二地址、单地址、零地址混合方式的指令格式。指令字长度分为：单字长、半字长、双字长三种形式，高档微型机中目前多采用 32 位长度的单字长形式。

(3) 寻址方式包括指令寻址方式、数据寻址方式。形成指令地址的方式，称为指令寻址方式，有顺序寻址和跳跃寻址两种。形成操作数地址的方式，称为数据寻址方式。操作数可放在专用寄存器、通用寄存器、内存和指令中。数据寻址方式有：立即寻址、直接寻址、间接寻址、寄存器寻址、寄存器间接寻址、相对寻址、基址寻址、变址寻址、块寻址、段寻址等多种。按操作数的物理位置不同，有 RR 型和 RS 型。前者比后者执行的速度快。堆栈是一种特殊的数据寻址方式，采用“先进后出”原理。按实现方法不同，分为硬堆栈和软堆栈。

(4) 不同机器有不同的指令系统，一个较完善的指令系统应当包含数据传送类指令、算术运算类指令、逻辑运算类指令、程序控制类指令、I/O 类指令、字符串类指令、系统控制类指令。

(5) RISC 指令系统是 CISC 指令系统的改进，它的最大特点是：①指令条数少；②指令长度固定，指令格式和寻址种类少；③只有取数/存数指令访问存储器，其余指令的操作均在寄存器之间进行。

习题三

一、名词解释

1. 机器指令　2. 操作码　3. 操作数　4. 指令系统
5. 有效地址　6. 寻址方式　7. 指令字长　8. CISC
9. RISC

二、选择题

1. 指令系统中采用不同寻址方式的主要目的是(　　)。
   A. 实现存储程序和程序控制
   B. 缩短指令长度,扩大寻址空间,提高编程灵活性
   C. 可以直接访问外存
   D. 提供扩展操作码的可能并降低指令译码难度

2. 下列说法中不正确的是(　　)。
   A. 机器语言和汇编语言都是面向机器的,它们和具体机器的指令系统密切相关
   B. 指令的地址字段指出的不是地址,而是操作数本身,这种寻址方式称为立即寻址
   C. 堆栈是存储器的一部分,也可以通过地址访问
   D. 机器中的寄存器和存储单元是统一编址的

3. 某计算机有16个通用寄存器,采用32位定长指令字,操作码字段(含寻址方式位)为8位,Store指令的源操作数和目的操作数分别采用寄存器直接寻址和基址寻址方式,若基址寄存器可使用任一通用寄存器,且偏移量用补码表示,则Store指令中偏移量的取值范围是(　　)。(2014年全国硕士研究生入学统考计算机学科专业试题)
   A. －32 768～＋32 767　B. －32 767～＋32 768
   C. 65 536～＋65 535　D. －65 535～＋65 536

4. 假设变址寄存器R的内容为1000H,指令中的形式地址为2000H;地址1000H中的内容为2000H,地址2000H中的内容为3000H,地址3000H中的内容为4000H,则变址寻址方式下访问到的操作数是(　　)。(2013年全国硕士研究生入学统考计算机学科专业试题)
   A. 1000H　B. 2000H　C. 3000H　D. 4000H

5. 偏移寻址通过将某个寄存器内容与一个形式地址相加而生成有效地址。下列寻址方式中,不属于偏移寻址方式的是(　　)。(2011年全国硕士研究生入学统考计算机学科专业试题)
   A. 间接寻址　B. 基址寻址　C. 相对寻址　D. 变址寻址

6. 某机器字长16位,主存按字节编址,转移指令采用相对寻址,由两个字节组成,第一字节为操作码字段,第二字节为相对位移量字段。假定取指令时,每取一个字节PC自动加1。若某转移指令所在主存地址为2000H,相对位移量字段的内容为06H,则该转移指令成功转移以后的目标地址是(　　)。(2009年全国硕士研究生入学统考计算机学科专业试题)
   A. 2006H　B. 2007H　C. 2008H　D. 2009H

7. 下列关于 RISC 的叙述中，错误的是（　　）。（2009 年全国硕士研究生入学统考计算机学科专业试题）

A. RISC 普遍采用微程序控制器

B. RISC 大多数指令在一个时钟周期内完成

C. RISC 的内部通用寄存器数量相对 CISC 多

D. RISC 的指令数、寻址方式和指令格式种类相对 CISC 少

三、综合题

1. 一个机器的指令系统通常应满足哪几方面的要求？

2. 常用机器指令格式有哪些？

3. 指令操作码主要有哪两种编码格式？

4. 指令操作通常分哪几类？

5. 如何减少一条指令中给出的地址数？又如何减少指令中表示一个地址码的位数？

6. 一台机器的指令系统为什么要提供多种寻址方式？

7. 什么是立即数寻址？立即数寻址中的“立即数”一般存放在哪里？

8. 简述相对寻址、变址寻址及基址寻址的原理，这三种寻址方式有哪些相同点和不同点？

9. 机器的堆栈寻址方式常作何用途？

10. 设某指令系统指令定长 12 位，每个地址段 3 位，试设计一种方案，使该指令系统有 4 条三地址指令，8 条二地址指令，180 条单地址指令。

11. 假设某计算机指令长度为 20 位，具有双操作数、单操作数、无操作数三类指令格式，每个操作数地址规定用 6 位表示，现已设计出 m 条双操作数指令，n 条无操作数指令。

试问：

（1）若采用定长 8 位操作码字段，则这台计算机最多可以设计出多少条单操作数指令？

（2）若操作码字段长度不固定，则这台计算机最多可以设计出多少条单操作数指令？

12. 某微型机指令格式如图 3-26 所示。

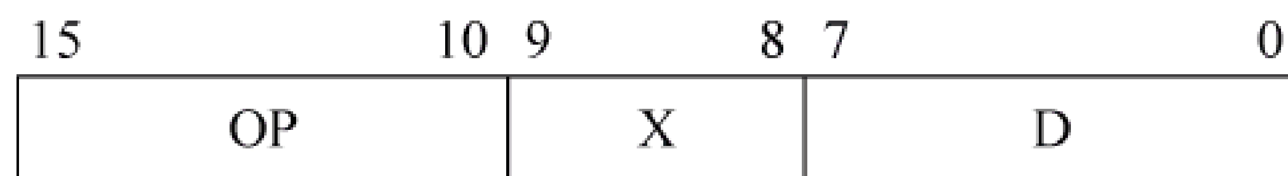


图 3-26　微型机指令格式

格式中 D 为位移量，X 为寻址方式特征值：

X＝00，直接寻址；

X＝01，用变址寄存器 R_1 进行变址；

X＝10，用变址寄存器 R_2 进行变址；

X＝11，相对寻址。

设(PC)＝1234H，(R_1)＝0037H，(R_2)＝1122H，（H 代表十六进制数），请确定以下指令的有效地址：(1)4420H；(2)2244H；(3)1322H；(4)3521H；(5)6723H。

13. 某指令系统指令定长 16 位，指令格式如图 3-27 所示，试分析指令格式及寻址方式的特点。

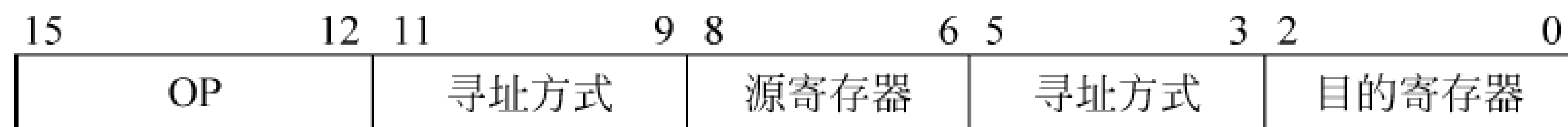


图 3-27 指令格式 1

14. 一种单地址指令格式如图 3-28 所示，其中 I 为间接特征，X 为寻址模式，D 为形式地址。I、X、D 组成该指令的操作数有效地址 EA。设 R 为变址寄存器，R_1 为基址寄存器，PC 为程序计数器，请在表 3-9 中第一列位置填入适当的寻址方式名称。

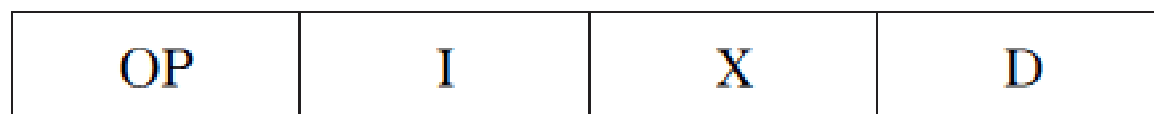


图 3-28 指令格式 2

表 3-9 填入寻址方式

寻址方式名称	I	X	有效地址 EA
(1)	0	00	EA=D
(2)	0	01	EA=(PC)+D
(3)	0	10	EA=(R)+D
(4)	0	11	EA=(R_1)
(5)	1	00	EA=(D)
(6)	1	11	EA=((R_1)+D),D=0

15. 一个处理器具有如图 3-29 所示的指令格式。

图 3-29 指令格式 3

其格式表明有 8 个通用寄存器(长度 16 位)，X 为指定的寻址模式，主存最大容量为 256KB。

(1) 假设不用通用寄存器也能直接访问主存的每一个操作数，并假设操作码域 OP=6 位，请问地址码域应该分配多少位？指令字长度应有多少位？

(2) 假设 X=11 时，指定的那个通用寄存器用作基址寄存器，请提出一个硬件设计规则，使得被指定的通用寄存器能访问 1MB 主存空间中的每一个单元。

16. (2013 年全国硕士研究生入学统考计算机学科专业试题)某计算机采用 16 位定长指令字格式，其 CPU 中有一个标志寄存器，其中包含进位/借位标志 CF、零标志 ZF 和符号标志 NF。假定为该机设计了条件转移指令，其格式如图 3-30 所示。

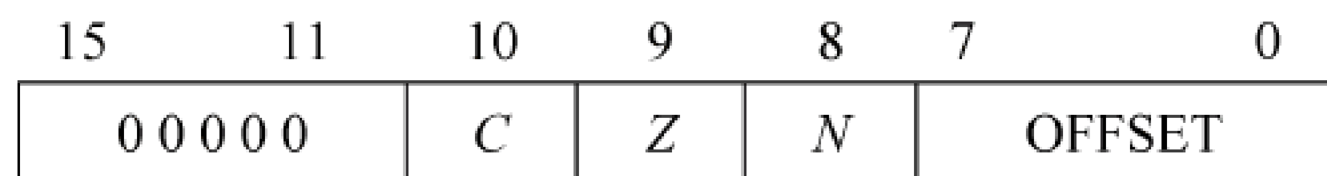


图 3-30 指令格式 4

其中，00000 为操作码 OP；C、Z 和 N 分别为 CF、ZF 和 NF 的对应检测位，某检测位为 1 时表示需检测对应标志，需检测的标志位中只要有一个为 1 就转移，否则不转移，若

$C=1, Z=0, N=1$，则需检测 CF 和 NF 的值，当 CF=1 或 NF=1 时发生转移；OFFSET 是相对偏移量，用补码表示。转移执行时，转移目标地址为(PC)+2+2×OFFSET；顺序执行时，下条指令地址为(PC)+2。请回答下列问题。

(1) 该计算机存储器按字节编址还是按字编址？该条件转移指令向后(反向)最多可跳转多少条指令？

(2) 某条件转移指令的地址为 200CH，指令内容如图 3-31 所示，若该指令执行时 CF=0，ZF=0，NF=1，则该指令执行后 PC 的值是多少？若该指令执行时 CF=1，ZF=0，NF=0，则该指令执行后 PC 的值又是多少？请给出计算过程。

15　　11	10	9	8	7　　0
00000	0	1	1	11100011

图 3-31　指令格式 5

(3) 实现“无符号数比较小于等于时转移”功能的指令中，C、Z 和 N 应各是什么？

第4章 存储系统组织与结构

现代计算机主要采用冯·诺依曼结构设计,这种计算机的特点是采用存储程序的思想,即将要运行的程序事先存放在存储器中,由CPU从存储器中取出来并执行。存储器的性能在很大程度上影响到机器的整体性能。

本章首先介绍存储器的组织、分类和分层结构,然后介绍计算机主存储器的组成与工作原理,最后介绍提高存储系统性能的交叉存储技术、高速缓冲存储器及虚拟存储器技术。

4.1 存储系统概述

本节首先让大家认识一下存储器的逻辑组织及CPU是如何对其进行操作和访问的,然后介绍计算机中都有哪些种类的存储器,最后介绍现代计算机普遍采用的存储层次。

4.1.1 存储器的组织

从物理上讲,存储器是由具有一定记忆功能的物理器件构成的。在不同历史阶段,随着存储技术的发展,计算机中采用了不同的物理器件构成其存储器,如20世纪60—70年代计算机中采用磁芯存储器作为计算机的主存,而目前计算机主存则普遍采用的是半导体存储器件。无论采用什么样的存储器件,从逻辑上讲,存储器可以看成由一个个存储单元构成的连续的地址空间。

如图4-1所示是存储器的逻辑组织结构图。该存储器共有2^{20}(1M)个存储单元,每个存储单元能存储8位二进制数据。每个单元都有一个20位的二进制存储单元地址(图4-1中采用十六进制表示),且相邻单元的地址是连续递增的,即自上而下,从最低地址00000H开始连续递增到最高地址FFFFFH。每个单元中存放的数据称为该单元的内容。例如,00000H单元的内容为30,00001H单元的内容为50。

地址	内容
00000H	30
00001H	50
00002H	
	⋮
FFFFFH	30

图4-1 存储器的逻辑组织

计算机CPU对存储器的操作主要有两种:一是读,二是写。所谓“读”,是指CPU将某个指定的存储单元的内容从存储器中取出,送入CPU或其他部件中处理;所谓“写”,是指CPU将其他某个部件中的数据写入指定的存储单元。CPU对存储器的读操作、写操作统称为CPU访存操作。

一般来讲,CPU与存储器之间的物理连接主要包括三组信号线:地址线、数据线和控

制线，如图 4-2 所示。

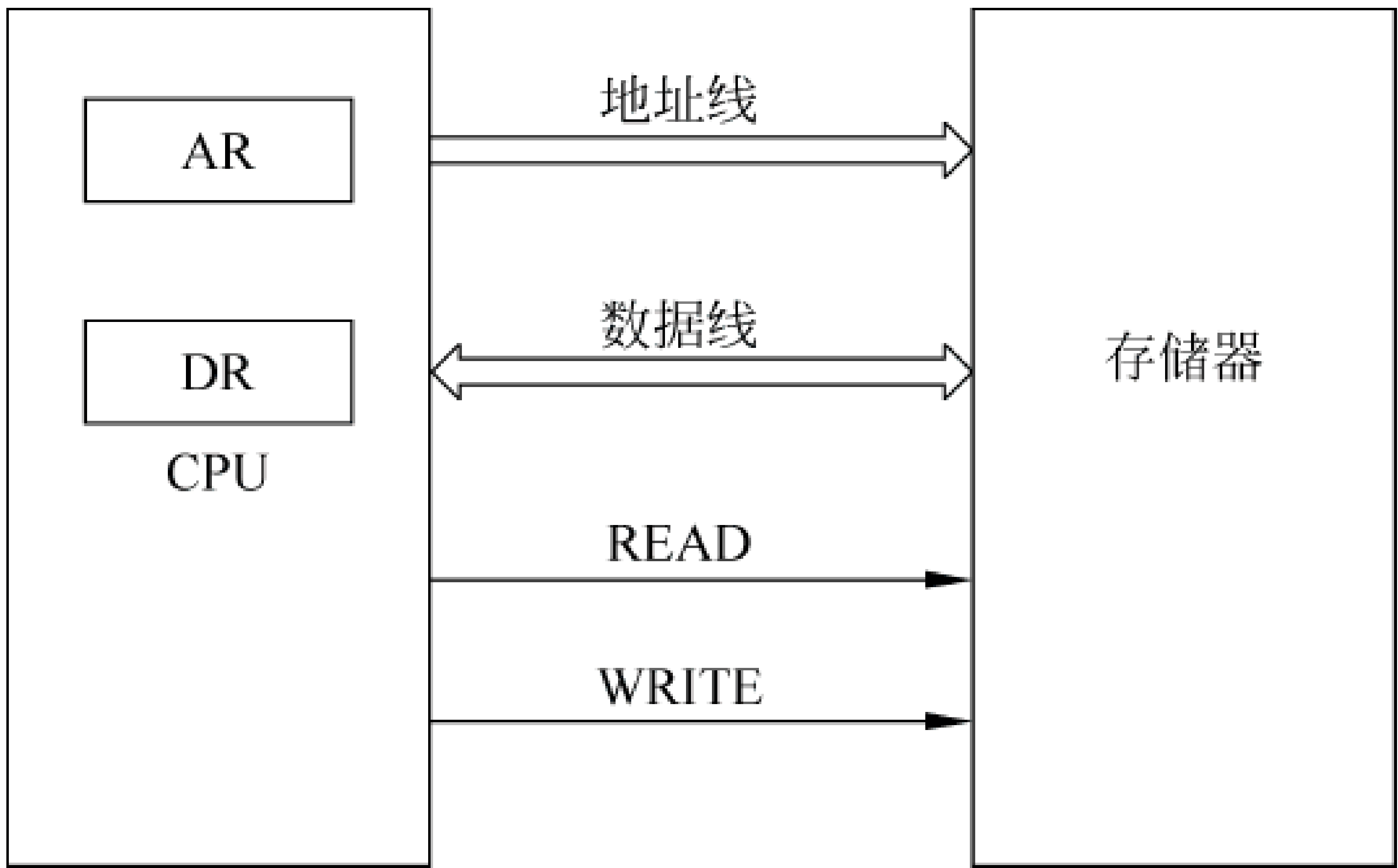
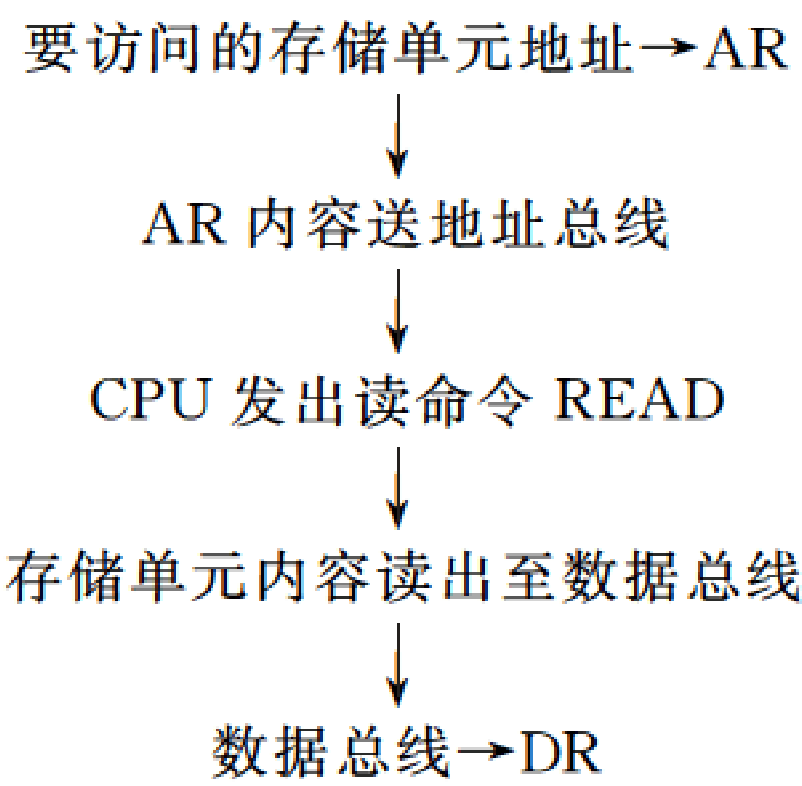


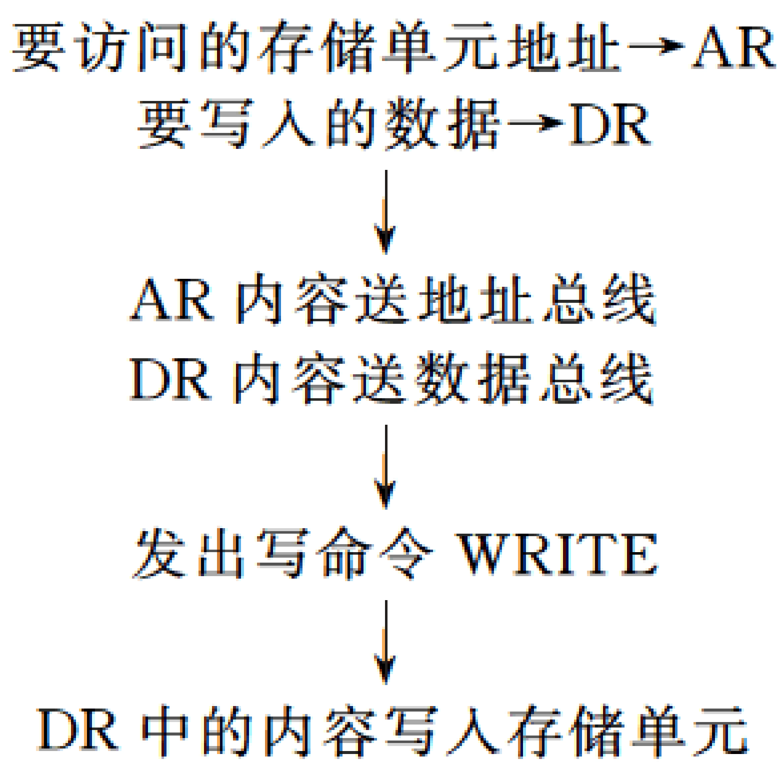
图 4-2　CPU 与存储器的连接

其中，AR 是地址寄存器，用于存放 CPU 要访问的存储单元的地址，CPU 对存储器的访问无论是读还是写，都首先要将要访问的存储单元的地址送入 AR 中；DR 是数据寄存器，用于存放 CPU 与存储器之间交换的数据。READ 和 WRITE 分别是 CPU 用于对存储器进行读或写的控制信号。CPU 对存储器进行读操作时，须给出 READ 控制信号；进行写操作时，须给出 WRITE 信号。

CPU 对存储器的读操作过程如下：



CPU 对存储器的写操作过程如下：



4.1.2　存储器的分类

在计算机发展的不同阶段，人们设计出了不同种类的存储器，如从最初的汞延迟线到磁芯存储器，再到今天的半导体存储器，一方面是为了满足计算机技术发展的需要，另一方面

也是与当时的器件技术及制作工艺水平相适应。

现代计算机存储器种类繁多，之所以这样，是因为不同种类的存储器可以满足不同应用的需要。从不同角度进行划分，会有不同种类的存储器。

按照存储器在计算机中的作用划分，主要包括计算机主存、计算机辅存和高速缓存。主存(Memory)又称为计算机内存，就是冯・诺依曼结构中的存储器，用于存储要执行的程序和处理的数据；辅存(Storage)又称为计算机外存，在现代冯・诺依曼机器中，程序在执行之前是以文件的方式存储在外存中的，当要运行某程序时，由操作系统将该程序从外存调入内存中；高速缓存(cache)是一种小容量、高速度的存储器，目前，在计算机的主板和CPU中均设置了高速缓存，设置高速缓存的目的是利用程序的局部性原理实现计算机的存储层次，提高CPU的访存速度，以匹配CPU和主存之间在速度上的差异。有关高速缓存的内容将在4.4节详细讲述。

按照存储器所使用的物理存储介质或材料划分，目前主流的存储器主要有三种：一是半导体存储器，它是用半导体材料制作成的存储器，通常是以存储器芯片的形式出现的，随着集成度的提高，单个存储器芯片的容量越来越大；二是磁介质存储器，使用聚酯塑料或金属薄片为基质，在其表面涂上磁性材料，采用磁记录原理存储信息，目前主流磁介质存储器主要有软盘存储器、硬盘存储器和磁带存储器等；三是光存储器，按光存储和读写原理制作成的存储器，如CD-ROM和可读写光盘等。半导体存储器与磁介质存储器和光存储器相比较，速度快，集成度高，价格也较高。在现代计算机中，半导体存储器主要用作计算机的主存，而磁介质存储器和光存储器则用作计算机辅存。

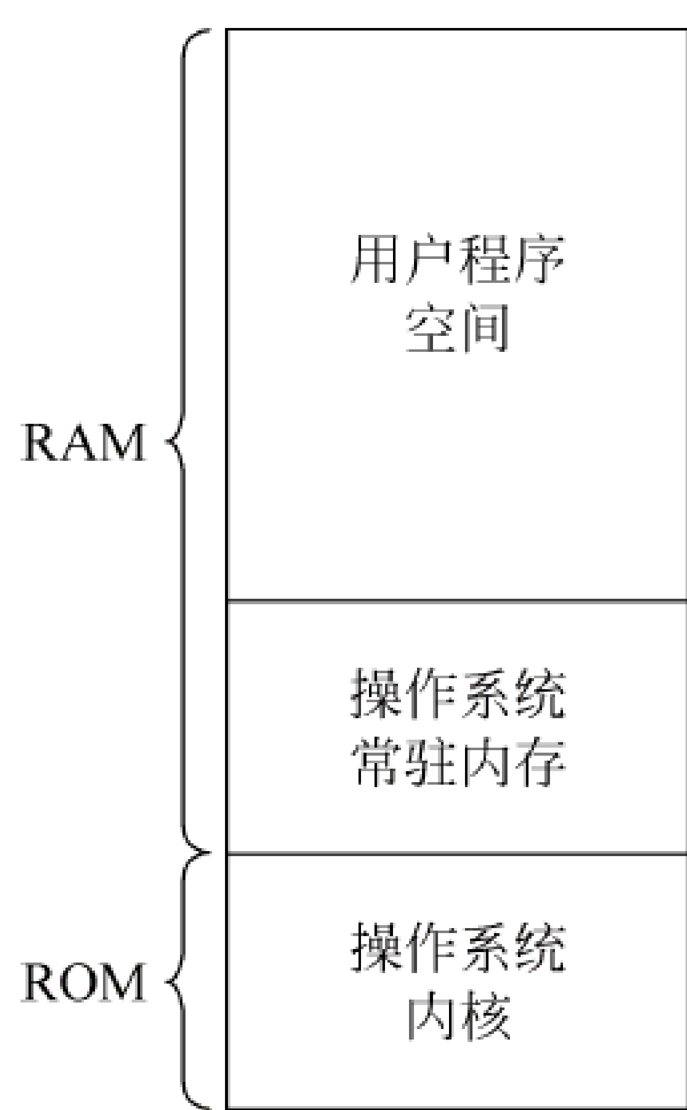


图4-3　计算机内存分布

按照存储器的读写功能划分，主要有两种：一是随机存取存储器(Random Access Memory，RAM)，可以对存储单元按地址随机存取；二是只读存储器(Read Only Memory，ROM)，在正常工作条件下，对单元内容只可读不可写。这两类存储器均属于半导体存储器，是构成计算机主存的主要存储介质，而其中RAM在计算机中用于存储操作系统的常驻内存部分和用户程序，ROM则用于存储操作系统的内核。计算机内存分布如图4-3所示。

除上述分类外，存储器还有易失性和非易失性之分。所谓易失性是指写入存储器中的内容在通电情况下能够保存，一掉电则全部丢失；非易失性则是指写入存储器中的内容在不通电的情况下仍然能够保存。上述磁介质存储器、光存储器和半导体存储器中的ROM均属于非易失性存储器，而RAM则属于易失性存储器。人们可能都遇到过这样一种情况，当使用字处理软件进行一个文本编辑时，突然停电了，待再来电时，发现先前编辑输入的内容中，未保存的部分不见了。其实，当新建一个文件并利用字处理软件编辑文本时，字处理系统会为用户编辑的文件在内存(RAM部分)开辟一个缓冲区。每当保存文件时，字处理系统就会将缓冲区中的内容写入外存的文件中，这时即使机器掉电，保存在外存文件中的内容仍然保留着，因为外存是由非易失性存储器构成的。而如果没有保存，这时机器掉电，则缓冲区中的内容将全部丢失。

将以上类型的存储器总结如图4-4所示。

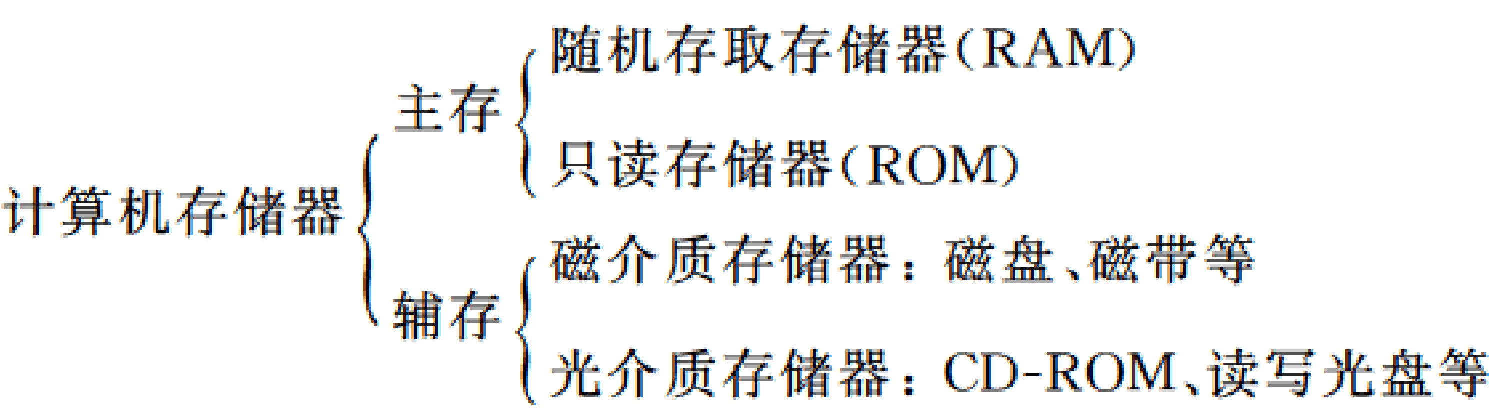


图 4-4　计算机存储器种类

在第 4 章，主要讲述构成计算机主存储器的半导体存储器，介绍半导体存储器的种类、单元电路的构成及工作原理和主存系统的设计等。磁介质存储器和光存储器等辅助存储器(以下简称辅存)的内容放在后面章节讲述。在计算机中，操作系统是将辅存作为设备来进行管理的。

4.1.3　存储器的分层结构

CPU 执行指令的过程实际上分为两个步骤：一是取指令，二是执行指令。CPU 与存储器之间是通过总线连接的，当 CPU 执行一条指令时，首先必须从存储器中将指令读出到总线上，再通过总线将指令传送到 CPU 内部，这一过程称为取指令；然后对指令进行译码分析，最后执行该指令，这一过程称为执行指令，如图 4-5 所示。

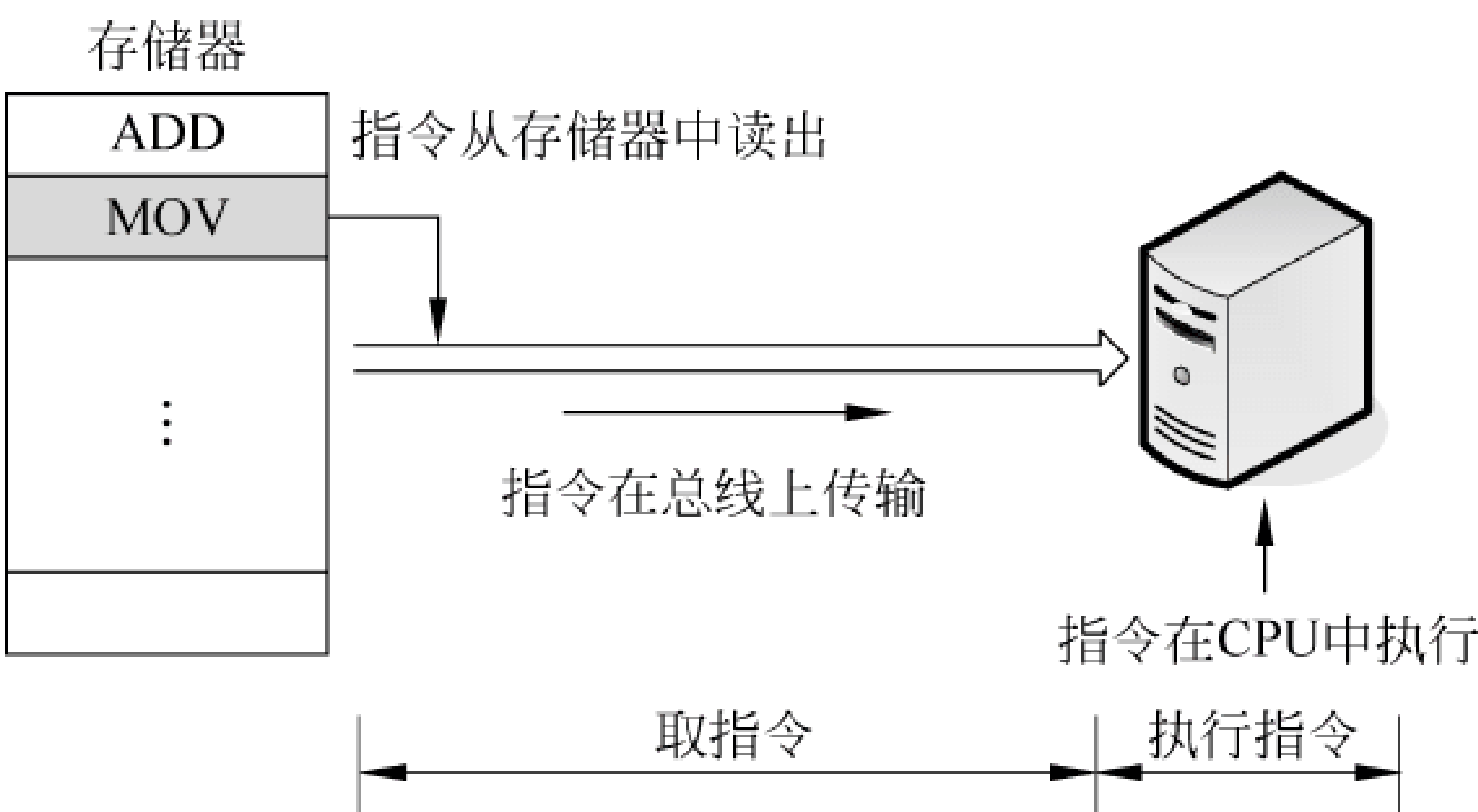


图 4-5　指令的执行过程

假设指令从存储器中读出的时间为 t_A，指令在总线上传输的时间为 t_B，指令在 CPU 中执行的时间为 t_E。这样，一条指令的执行时间为 $t_A+t_B+t_E$，称为指令周期。t_B 的大小取决于总线传输速度，而 t_A 的大小取决于所配置的存储器的存取速度，这一时间实际上就是存储器的读周期时间，对某一存储器而言，这一时间是固定的。由此可以看出，计算机的性能是由组成它的各个功能部件的综合性能来体现的，光有高速的 CPU 还不够，还要有高速的存储器和高速的总线传输等。

因此，当用户在配置一台计算机时，对机器的存储器总是希望其容量越大越好，速度越快越好，同时还希望价格越便宜越好。但性能和价格却往往是一对相互矛盾的因素，性能高往往意味着价格高。当用户为了某种应用而不惜代价时，可以为机器配置大容量高速存储器。但作为面向一般用户的通用计算机，这样做则会令用户在其面前望而却步。那么是否有什么办法可以让用户做到用较低的价格配置到高速大容量的存储器呢？用户的需要就是计算机设计者努力的方向，现代计算机存储器普遍采用如图 4-6 所示的层次结构来实现上述目标。

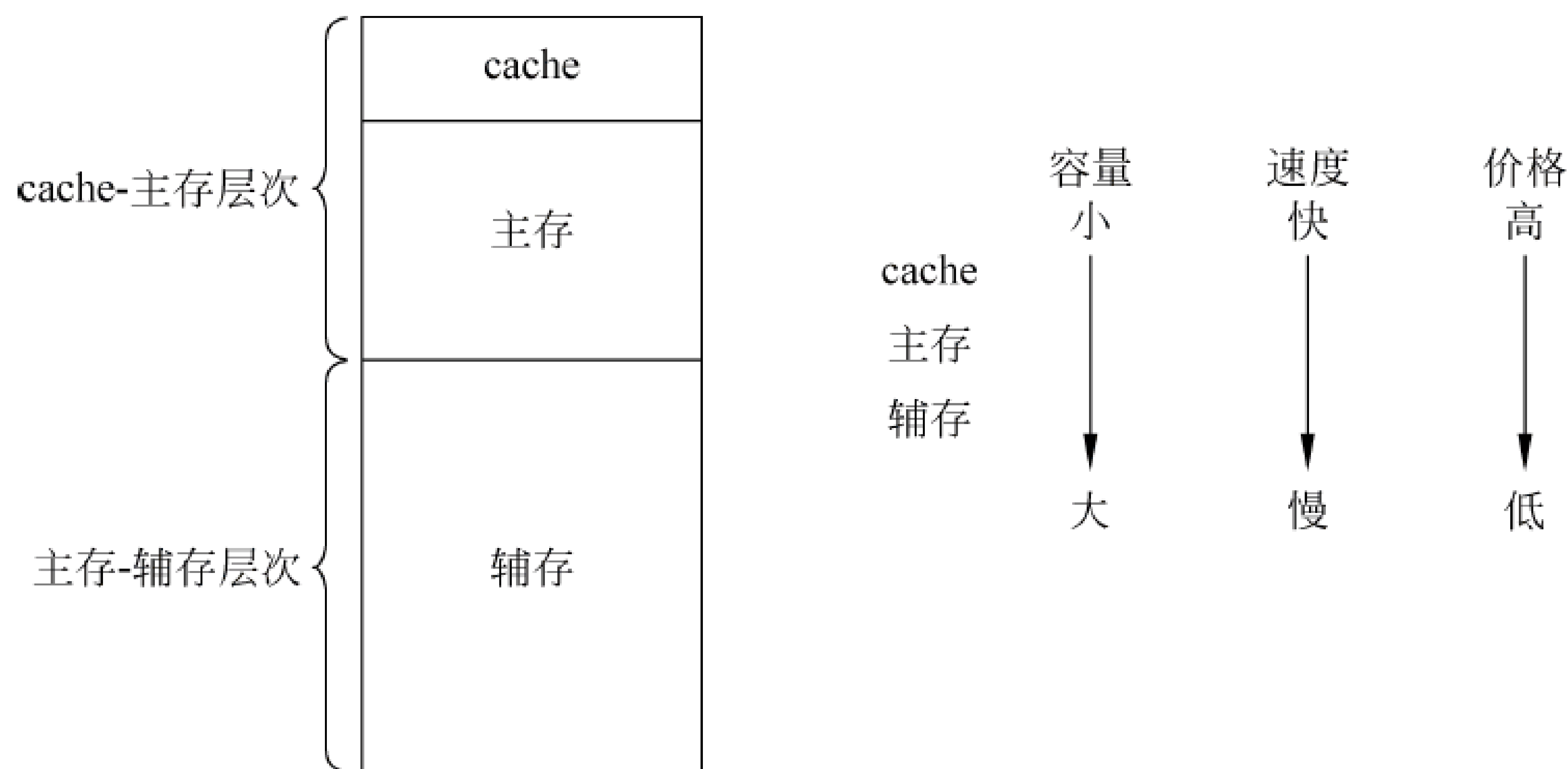


图 4-6 存储层次

存储器的层次包括高速缓冲存储器(cache)、主存和辅存,其中,cache 是由小容量(通常为几十 KB 到几十 MB)、高速的半导体存储器件构成的,当然这种器件的价格也相对较高;主存一般使用动态半导体存储器件构成,在速度上比 cache 慢,但容量更大,价格相对更低,现代机器根据应用要求的不同通常将主存配置到几百 MB 至几个 GB,一些对内存要求很高的场合可配置到几十 GB 甚至更大;辅存又称海量存储器,通常由磁介质存储器构成,其特点是容量大,速度慢,价格便宜。cache 又可以进一步分为片上 cache 和板上 cache。片上 cache 是指 CPU 内部配置的 cache,又称为一级 cache,板上 cache 是指计算机主板上配置的 cache,又称为二级 cache。一级 cache 比二级 cache 的容量更小,但速度更快。在有些机器中,还进一步将指令和数据分开,分别存放在指令缓存和数据缓存中。

存储器的这种分层结构将为用户的程序提供一个在容量方面相当于辅存容量,而在 CPU 的访问速度方面则接近于访问 cache 的速度的存储空间。如何能做到这一点,本书将分别在 4.4 节和 4.5 节进一步讲解。

4.2 半导体存储器

半导体存储器是目前构成计算机主存储器的主要存储介质。本节首先介绍半导体存储器的种类,然后介绍几种半导体存储器的单元电路的组成及工作原理,最后介绍如何设计计算机的主存储器。

4.2.1 半导体存储器的种类

计算机从所运行的程序种类上讲,主要分为操作系统程序和其他应用程序。操作系统是整个机器系统的软件核心,从构成上讲,它主要包括操作系统内核(Core)和操作系统的外壳(Shell),其中,内核起着引导机器系统启动和为应用程序提供输入输出服务和管理的作用(例如,DOS 操作系统的基本输入输出系统 BIOS)。由于操作系统内核无论机器是否通电都必须永久保存,因此在机器中通常使用非易失性存储器 ROM 存储,而操作系统的外壳和其他应用程序均可存储在易失性存储器 RAM 中,其中操作系统的外壳是操作系统常驻内存部分,通常是在机器启动时从辅存中调入内存中的,而用户程序是在运行时由操作系统

调入内存中的。

随机存取存储器（RAM）按构成其单元电路的不同，又分为静态随机存取存储器（SRAM）和动态随机存取存储器（DRAM）。这两种存储器相比较而言，SRAM速度更快，但片容量小，价格更贵，因此SRAM主要用作计算机的cache，而DRAM则用作计算机主存。

只读存储器ROM种类较多，主要包括传统的掩膜式ROM、一次可编程式ROM（PROM）、紫外线可擦除可编程式ROM（UVEPROM）和电可擦除可编程式ROM（E^2PROM）。前三种ROM在正常工作条件下只可读不可写，是真正意义上的只读存储器，而E^2PROM实际上可读可写，但与RAM相比较，最大的不同是：它们均是非易失性的。

掩膜式ROM是一种定制式ROM，生产厂家根据用户的需要专门制作掩膜版，将用户提交的程序或数据固定制作在芯片上。这种芯片的特点是，一旦制作好，芯片中的内容就无法进行修改。因此，这种芯片主要用于用户需大批量生产具有固定功能的ROM的场合使用，如在工业控制、家用电器等中均有应用。掩膜式ROM的优点是当用户大批量生产时成本相对较低，但缺点是使用不够灵活，内容的写入需提交厂家完成，而且用户需要大量订购才能使成本降低。

一次可编程式PROM（Programmable ROM）顾名思义是这种存储器可以进行一次写入或编程。PROM出厂时是空片，芯片的内容为全0或全1，用户可以根据自己的需要将编制好的程序或数据一次写入，一旦写入内容就无法再进行修改。PROM相对掩膜式ROM来说，使用更加灵活，用户可以根据自己的需要批量购买，也可以只购买几片。

紫外线可擦除可编程式（UV Erasable Programmable ROM，UVEPROM）是一种可重复改写的ROM，当用户要对芯片中的内容进行修改或重写时，需首先使用紫外线对芯片的擦除窗口进行照射（一般可放置在专门的紫外线光源下照射），经过一段时间后，芯片中的内容全部被擦除，成为空片，然后使用专门的编程器，将新的内容写入芯片中。由于太阳光中存在紫外线，因此，这种芯片在使用中应注意避免阳光的照射，通常的做法是用黑色的胶纸将芯片的擦除窗口贴起来。UVEPROM由于其使用的灵活性，应用较为广泛，例如，早些年微型计算机主板上的用于存储操作系统内核的ROM和计算机网卡上用于无盘引导的ROM芯片均使用了UVEPROM。UVEPROM的外观如图4-7所示。

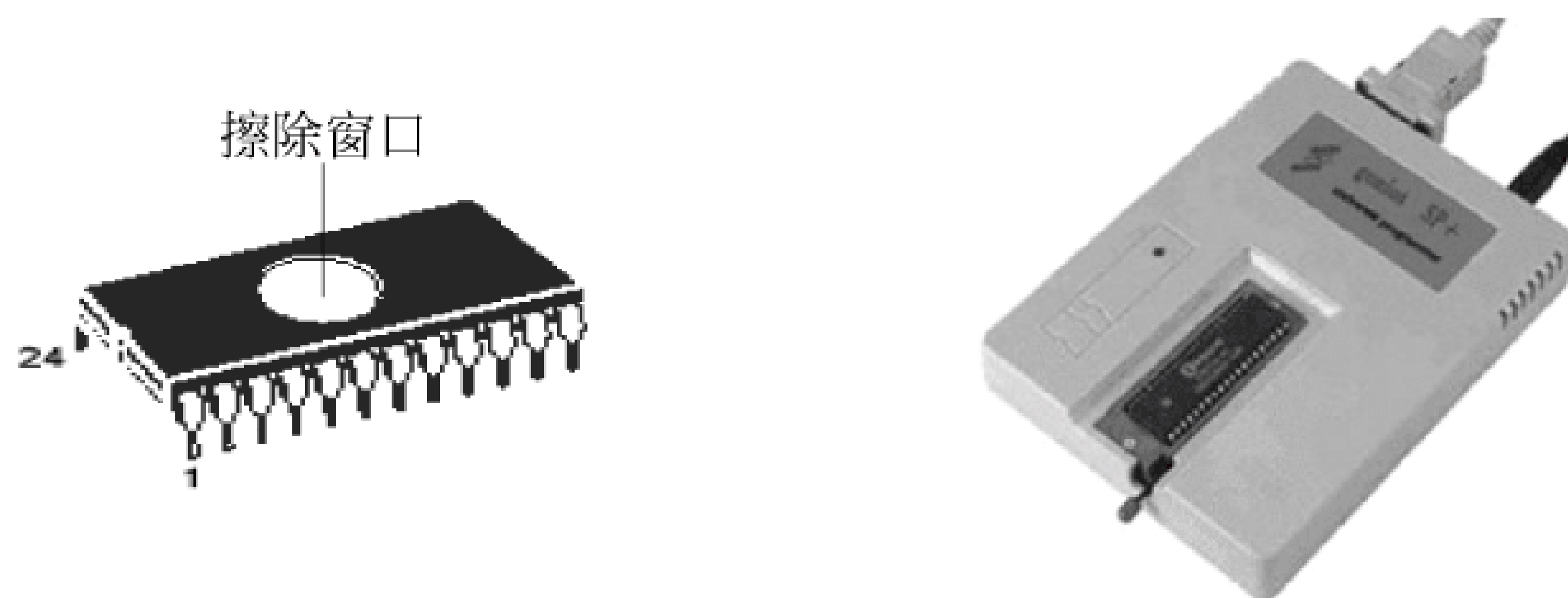


图4-7　UVEPROM和编程器

电可擦除可编程式（Electronic Erasable Programmable ROM，E^2PROM）也属于可重复改写的ROM。虽然UVEPROM相对PROM来说，可以多次改写，但其改写的条件却比较特殊，一般需要将其从工作的电路板上拔下来，使用专门的设备进行改写操作，这一方面一

般的用户无法做到，另一方面在有些应用场合，需要对芯片进行在线改写。电可擦除可编程式(E^2PROM)就是为满足这种应用而设计的，对 E^2PROM 可以像对 RAM 一样随机读写。因此，在使用上，E^2PROM 更加灵活和方便，其在计算机中的应用也更加广泛，如现代计算机主板上和计算机网卡上用于保存系统设置参数的存储器就是使用了 E^2PROM。

半导体存储器的种类总结如图 4-8 所示。

图 4-8 半导体存储器种类

4.2.2 半导体存储器的组成与工作原理

1. 静态 RAM 的组成及工作原理

半导体存储器是由一个个位单元电路组成，每个位单元电路存储一位二进制信息。静态 RAM 的位单元电路使用由三极管构成的触发器组成，利用触发器的状态存储 0、1 信息。如图 4-9 所示是一个 MOS 型六管静态 RAM 单元电路的构成图。

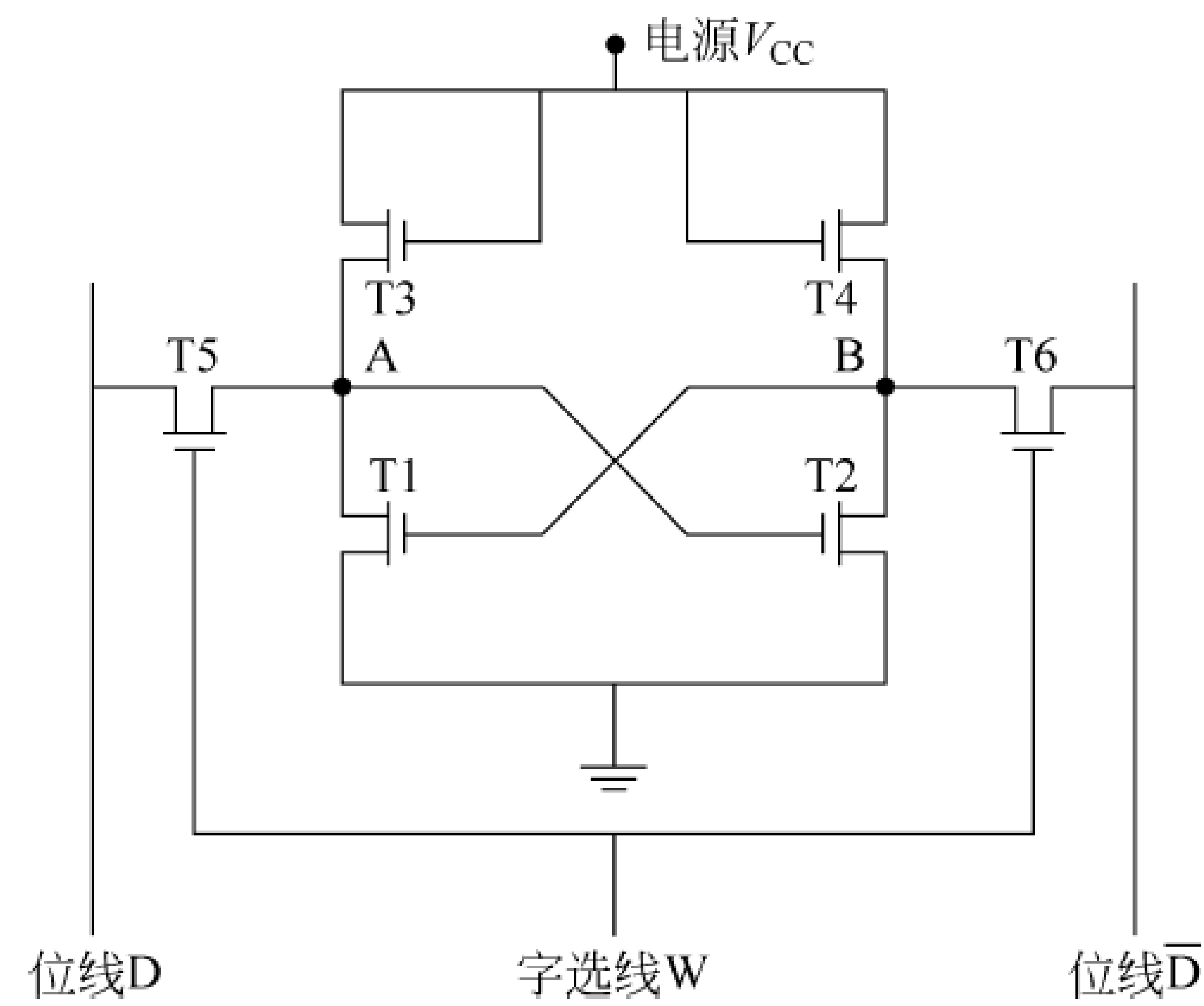


图 4-9 六管静态 RAM 单元电路

图 4-9 中，T1、T2 为存储管，任何时候，只要单元电路通电，T1、T2 总是处于两种状态之一：T1 导通、T2 截止或 T1 截止、T2 导通，正好可以利用这两种状态存储“1”或“0”。T3、T4 为负载管，分别起着 T1、T2 的极电阻的作用。T5、T6 为控制管，起着控制位单元与外界的接通和断开的作用。

假设定义：T1 导通、T2 截止时为存储了“1”；T1 截止、T2 导通时为存储了“0”，则其保持、读出和写入的工作原理如下：

保持：字选线 W 为低电平，则 T5、T6 管截止，使得 T1、T2 与外界电路切断，它们的状

态保持不变，从而所存储的信息能保持。

写入：字选线 W 为高电平，则 T5、T6 管导通。若欲写入“0”，则在 T5、T6 管的位线上分别加上高、低电位，A、B 点的电位也就分别为高、低电位，A 点的高电位使 T2 导通，B 点的低电位使 T1 截止，此即为“0”状态；若欲写入“1”，则在 T5、T6 管的位线上分别加上低、高电位，从而使 T1 导通、T2 截止，此即为“1”状态。

读出：字选线 W 为高电平，则 T5、T6 管导通。若原来存储的是“0”，则 A 点为高电位，A 经 T6 产生一个电流通路，即在 T6 位线上可检测到电流，如此读出了“0”；若原来存储的是“1”，则 B 点为高电位，B 经 T5 产生一个电流通路，即在 T5 位线上可检测到电流，如此读出了“1”。

以上位单元是构成静态 RAM 的基础。首先由一个个位单元组合在一起构成一个位单元阵列，再加上用于选择位单元的译码电路和读写控制等外围电路，最后进行封装，便构成一个静态 RAM 芯片。如图 4-10 所示是一个静态 RAM 芯片的组成结构图。

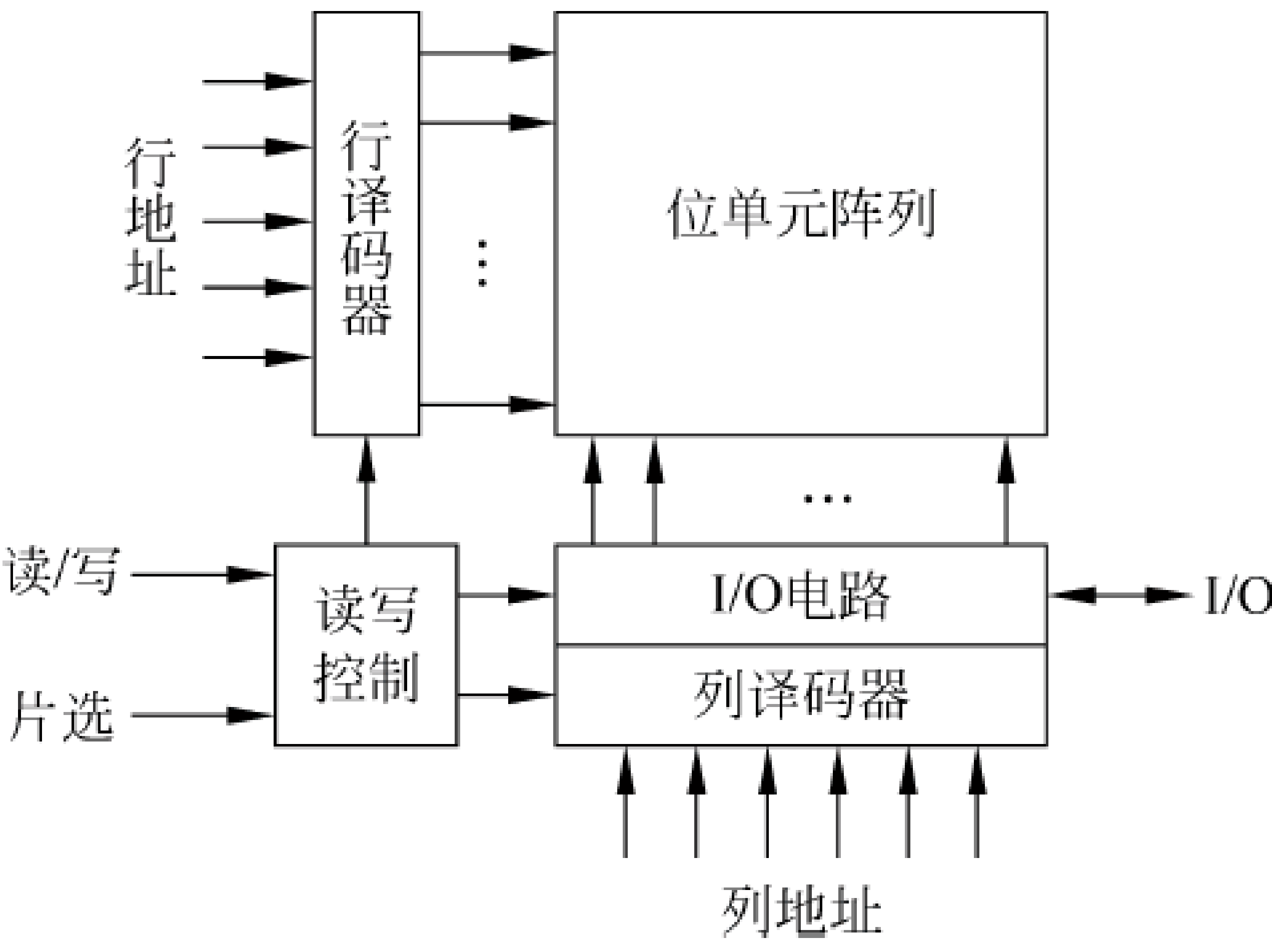


图 4-10　静态 RAM 组成结构图

位单元阵列通常排列成二维矩阵的形式，对位单元访问的地址分成行地址和列地址两部分，行、列地址分别通过行、列译码器产生行选信号和列选信号，选中相应的位单元，读/写信号控制对所选中单元的读/写操作。

2. 动态 RAM 的组成及工作原理

动态 RAM 的位单元电路与静态 RAM 相比有很大的不同，静态 RAM 的位单元电路是依靠由三极管构成的触发器的状态来存储 0、1 信息的，而动态 RAM 的位单元电路则是依靠三极管上的极电容的电荷的有无来存储 0、1 信息的。如图 4-11 所示是一个单管动态 RAM 位单元电路。

图 4-11 中，MOS 型晶体管 T 的栅极和漏极分别连接字选线 W 和位线 D，而源极上则制作了一个电容 *C*。该位单元电路正是依靠电容 *C* 上电荷的状态来存储 0、1 信息的。由于电容能存储一定范围内的任何电荷值，为此需设置一个阈值，以确定高于某阈值时为满电荷状态，存储的是“1”，而低于某阈值时为无电荷状态，存储的是“0”。其保持、读出和写入的工作原理如下：

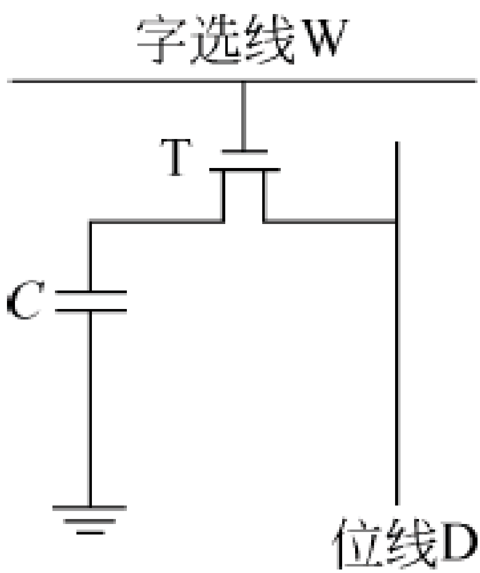


图 4-11　动态 RAM 组成结构图

保持：字选线 W 为低电平，则 T 截止，使得电容 C 与外部位线 D 之间的连接断开，C 上电容的状态保持不变，从而所存储的信息能保持。

写入：字选线 W 为高电位，则 T 导通。若写入“1”，则在位线 D 上加高电位，对 C 进行充电，使电容 C 充满电荷，从而处于“1”状态；若写入“0”，则在 D 线上加低电位，C 通过位线 D 对外进行放电，从而变为“0”状态。

读出：字选线 W 线为高电位，T 导通。若原来存储的是“1”，则 C 通过 T 管向位线 D 放电，在 D 线上可检测到电流流出，如此读出了“1”；若原来存储的是“0”，则在位线 D 上检测不到电流流出，如此读出了“0”。

从以上的读出过程可以看出，动态 RAM 的读出是一种破坏性读出，即如果位单元原来存储的是“1”，则经读出操作后，其状态变成了“0”。因此，动态 RAM 单元电路在经过读出操作后需要再进行“回写”操作。

另外，动态 RAM 单元电路的保持状态无法做到像静态 RAM 那样理想地保持。动态 RAM 在保持状态下，字选线为低电位，T 管截止。理论上电容 C 与外界的连接被断开，C 上的电荷状态能保持不变。但实际上，电容 C 与位线 D 之间的通路上总会有漏电流的存在，使得原来充满电荷存储“1”信息的 C 会逐渐对外放电，电荷随之丢失，从而也使其状态变成了“0”。为解决动态 RAM 单元电路存在的这一问题，一种有效的解决办法就是定时地对单元电路进行刷新，刷新的方式是周期性地对动态 RAM 单元电路进行逐行地无输出的“读出-回写”操作，使每个存储单元电路在信息丢失前进行及时的恢复。动态 RAM 的刷新方法主要有两种：一是在动态 RAM 内部设计刷新电路，存储器能自身完成刷新工作；二是在设计主存储器系统时，额外设计刷新外围电路，由 CPU 使用专门的刷新周期对动态 RAM 进行刷新操作。

从上次对整个存储器刷新结束到下次对整个存储器全部刷新一遍为止的时间间隔称为刷新周期，一般为 2ms。常用的刷新方式有三种：集中式刷新、分散式刷新和异步刷新。

1）集中式刷新

在整个刷新间隔内，前一段时间用于正常的读/写操作，而在后一段时间停止读/写操作，逐行进行刷新。集中刷新是在规定的一个刷新周期内，对全部存储单元集中一段时间逐行进行刷新，此刻必须停止读/写操作。这种方法的缺点在于出现了访存“死区”，对高速高效的计算机系统工作是不利的。

2）分散式刷新

分散式刷新是指对每行存储单元的刷新分散到每个读/写周期内完成。把存取周期分成两段，前半段用来读写或维持，后半段用来刷新。这种刷新克服了集中刷新出现的“死区”缺点，但它并不能提高整机的工作效率。

3）异步刷新

异步刷新方式是前两种方式的结合，为了真正提高整机的工作效率，应该采用集中与分散相结合的方式，既克服出现“死区”，又充分利用最大刷新间隔为 2ms 的特点。

动态 RAM 的芯片构成与静态 RAM 的相同，所不同的是位单元电路的构成和存储机制。从以上两者的单元电路也可以看出，动态 RAM 的单元电路构成更简单，因此，单片动态 RAM 芯片的容量更大。另外，由于动态 RAM 的读出需要回写和刷新，因此，动态 RAM 的速度比静态 RAM 的慢。当然，静态 RAM 的制作成本相对较高。基于动态 RAM 和静态

RAM 的这些特点，动态 RAM 主要用于构成计算机主存储器，而静态 RAM 主要用于构成快速的 cache。

3. ROM 的组成及工作原理

根据 ROM 种类的不同，位单元电路的组成也有所不同。

对于掩膜式 ROM 来讲，它的基本位单元电路使用二极管或晶体管做成，且根据用户程序或数据的需要分别制作，做管子的地方表示存储了"1"，没做管子的地方表示存储了"0"。由于每一位上的管子是做死的，因此，其内容是无法修改的。

图 4-12 给出了一个掩膜式 ROM 的组成示意图，其工作原理是：经行、列地址译码，各有一条地址线被选中(行方向来高电位，列方向来低电位)，交叉处则为被选中的单元。若交叉处有三极管，则被导通，输出处为低电位，表示读出了 1；若交叉处无三极管，则被截止，输出处为高电位，表示读出了 0。

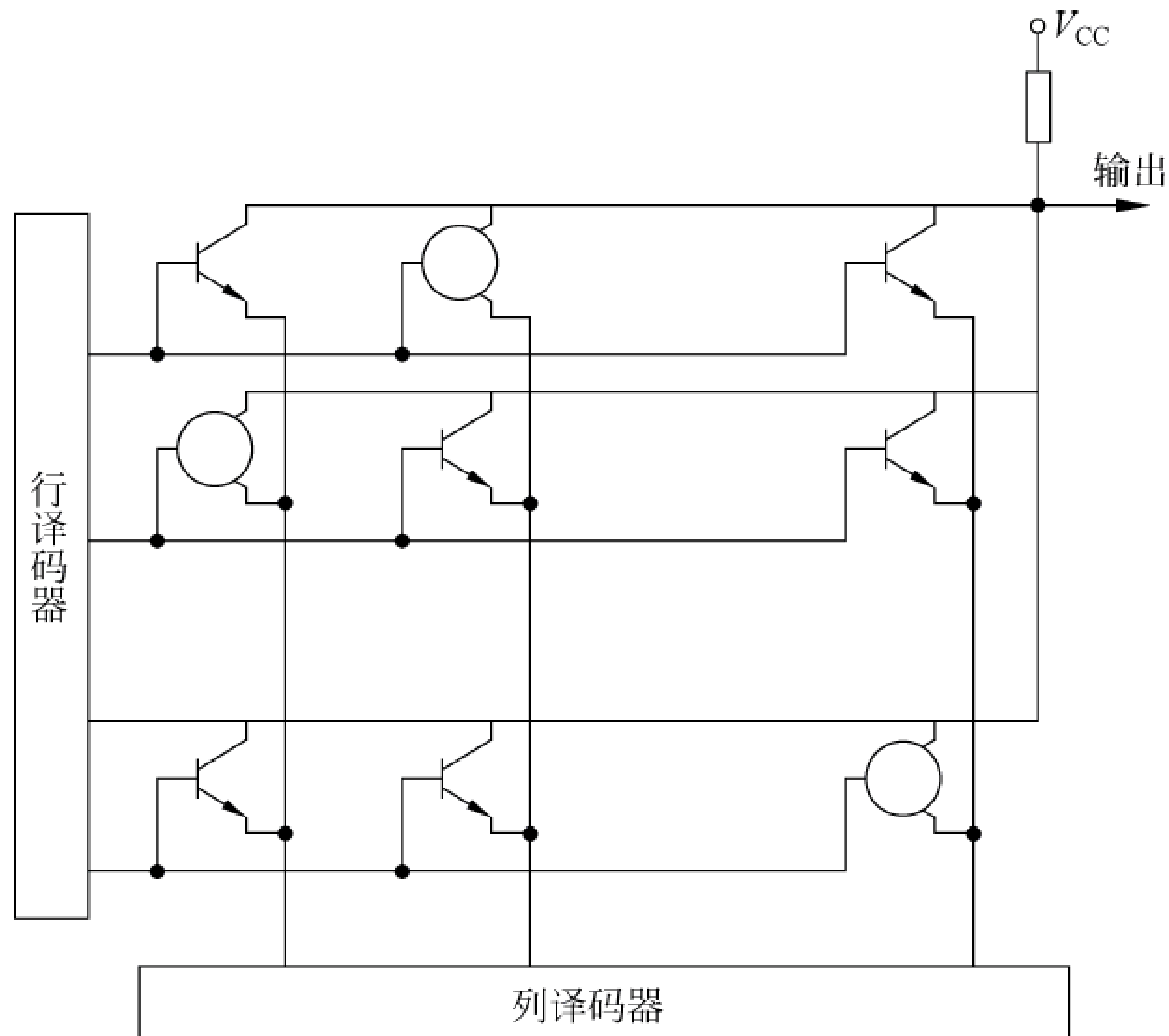


图 4-12　掩膜式 ROM 组成的示意图

而对于 PROM 来讲，其每一位位单元上均制作了晶体管，只是这种管子非常特殊，经过一次写入烧制后，其状态发生变化，且无法还原，故只具有一次可编程功能。有两种用于制作 PROM 位单元的晶体管：一种是熔断丝型，这种 PROM 是在每一个晶体管的发射极上串接一个熔断丝，芯片出厂时熔断丝全部为正常导通状态(全 1 状态)，当要对某一位改写时，只需将这一位上晶体管的熔断丝烧断，则晶体管将永远处于截止状态，表示存储了 0；另一种是结击穿型，这种 PROM 是在每一个晶体管的发射极上串接一个反向二极管，芯片出厂时反向二极管使每个晶体管处于截止状态(全 0 状态)，当要对某一位改写时，只需将这一位上的二极管击穿，则晶体管将永远处于导通状态，表示存储了 1。PROM 由于只能进行一次改写，因此目前已很少使用。

现在取代 PROM 的主要是 UVEPROM 和 E^2PROM。有关它们的存储原理，由于篇幅的限制，此不再赘述，读者有兴趣可参考相关资料。

Flash Memory 称为闪速存储器(简称闪存),是在 EPROM 的基础上发展起来的一种新型半导体存储器,于 1983 年推出,1988 年逐渐商品化,进入 20 世纪 90 年代中后期开始逐步得到广泛应用。从功能特性上讲,它可以归属于 ROM,因为它具有非易失性的特点。但从目前计算机的使用上讲,它是作为计算机外部存储器来使用的。在速度、集成度和成本上,Flash Memory 与传统的用于计算机主存 ROM 的芯片相近;而在使用上,Flash Memory 可以像磁盘等计算机辅助存储器一样通过一定的接口与主机相连,但却比磁盘等更方便,它无须配备像对磁盘进行读写操作的机电设备。因此,Flash Memory 自诞生之日起就得到了广泛的应用,除可用作计算机外存外(U 盘),一些家用电器产品,如手机、数码相机、MP3 等中的存储器均使用的是 Flash Memory。

传统的"CPU-主存-辅存"工作模式是将可执行程序先存放在辅存中,当要运行时,由 OS 将其从辅存调入主存中。OS 承担了对辅存和主存的空间分配及管理等工作,系统开销较大。而若采用新型的"CPU-Flash Memory"工作模式,Flash Memory 相当于传统主存和辅存的合二为一,CPU 可直接运行已经存放在闪存中的可执行程序。这样既可简化系统的硬件设计,又可大大降低系统的开销。随着 Flash Memory 成本的降低和集成度的提高,将来的某一天,"CPU-Flash Memory"工作模式有可能取代"CPU-主存-辅存"模式,这将引起计算机系统结构的巨大变化,这种变化将使得计算机的使用更加灵活和方便。

知识拓展

新型动态存储器 SDRAM 和 DDR、DDR2 和 DDR3

SDRAM 的全称为同步动态随机存取存储器(Synchronous Dynamic Random Access Memory),是一种新型动态存储器。传统的 DRAM 是异步工作的,CPU 向存储器发送地址和控制信号后,需等待存储器进行内部操作(地址译码、读出信号放大、输出缓冲等),从而影响了系统性能。而 SDRAM 与 CPU 之间是同步工作的,在进行存储器内部操作的同时,允许 CPU 做其他事情,而无须空等待。

DDR 全称为双倍速率同步动态随机存取存储器,即 DDR SDRAM(Double Date Rate Synchronous Dynamic Random Access Memory),是一种在 SDRAM 基础上发展起来的具有双数据速率的新型动态存储器,由三星公司于 1996 年提出,由日本电气、三菱、富士通、东芝、日立、德州仪器、三星及现代 8 家公司协议订立内存规格,并得到了 AMD、VIA 与 SiS 等主要芯片组厂商的支持。SDRAM 是在每一个时钟周期的时钟的上升沿进行一次数据传输;而 DDR 则是在每一个时钟周期内传输两次数据,即分别在时钟的上升沿和下降沿各传输一次数据,因此称为双倍速率同步动态随机存取存储器。DDR 内存可以在与 SDRAM 相同的总线频率下达到更高的数据传输率。目前很多计算机都使用了 DDR 存储器芯片作为其主存储器。

DDR2/DDR Ⅱ(Double Data Rate 2)SDRAM 是由 JEDEC(电子设备工程联合委员会)进行开发的新生代内存技术标准,它与上一代 DDR 内存技术标准最大的不同就是,虽然同是采用了在时钟的上升/下降延同时进行数据传输的基本方式,但 DDR2 内存却拥有两倍于上一代 DDR 内存预读取能力(即 4b 数据读预取)。换句话说,DDR2 内存每个时钟能够以 4 倍外部总线的速度读/写数据,并且能够以内部控制总线 4 倍的速度运行。

DDR3 是一种电脑内存规格。它属于 SDRAM 家族的内存产品,提供了相较于 DDR2 SDRAM 更高的运行效能与更低的电压,是 DDR2 SDRAM(4 倍资料率同步动态随机存取

内存)的后继者(增加至8倍),也是现时流行的内存产品。

4.2.3　主存储器的设计

半导体存储器是构成现代计算机主存储器的主要存储介质,在进行计算机主存储器设计时,主要考虑以下几方面的因素:

(1) 存储器芯片的选择,半导体存储器芯片种类繁多,在选择存储器芯片构成机器主存储器时,应根据需要合理选择。

(2) CPU与存储器的速度匹配,机器的性能是由多方面的因素决定的,其中CPU访存速度是影响机器性能的关键因素之一,高性能CPU需要高速的存储器相匹配。

(3) 存储器与CPU的信号连接,主要包括数据信号线、地址信号线和控制信号线的连接等。

1. 半导体存储器芯片

半导体存储器芯片的性能主要体现在两个方面:一是芯片的容量,二是芯片的存取速度。存储器芯片的容量可以表征为以下形式:

容量=字数×位数

其中,字数表示存储器芯片所具有的字单元数,而位数则表示每一个字单元所具有的位单元数。例如,静态RAM芯片2114的容量为1K×4b,则表示该芯片共有1024个字单元,每个字单元的位数为4b,也就是说,对2114的访问,一次可同时读/写4b。再如,只读存储器ROM芯片2716的容量为2K×8b,则表示该芯片共有2048个字单元,每个字单元的位数为8b,也就是说,对2716的访问,一次可以并行读出8b。

半导体存储器芯片是通过引出脚与外部连接的。引出脚主要包括4类:数据引脚、地址引脚、控制引脚和电源及接地引脚。图4-13(a)、图4-13(b)、图4-13(c)分别给出的是三种芯片256Kb的SRAM、16Mb的DRAM和8Mb的EPROM的引脚图。

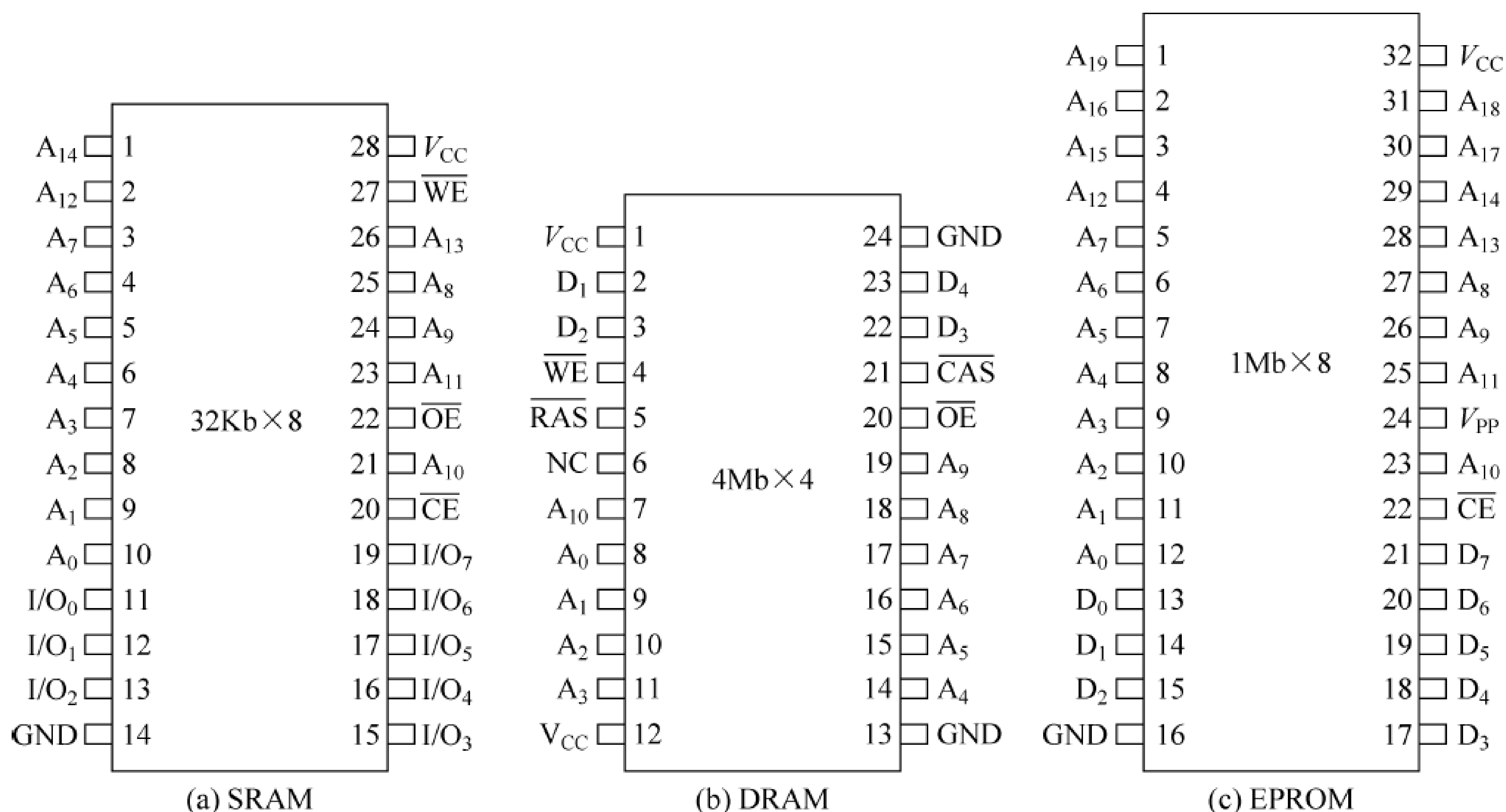


图4-13　芯片引脚图

对图 4-13(a)的 SRAM 芯片来说，各引脚的功能如下。

(1) 引脚 $A_0 \sim A_{14}$：15 条地址信号线，用于访问 $2^{15}=32K$ 的字单元，该地址是 CPU 访存时给出的，所以对存储器芯片来说，地址线是单向输入的。

(2) 引脚 $I/O_0 \sim I/O_7$：8 条数据信号线。CPU 对存储器进行读操作时，数据的流向是从存储器到 CPU 的；CPU 对存储器进行写操作时，数据的流向是从 CPU 到存储器的，所以数据信号线为输入输出双向。

(3) 引脚 $\overline{CE}$：为片选控制信号线。当 CPU 访存时，必须在这个引脚上加载一个有效信号，才能使存储器芯片工作。

(4) 引脚 $\overline{WE}$：读写控制信号。当 CPU 对芯片进行写操作时，在该引脚上加载一个低电平信号；当对芯片进行读操作时，在该引脚上加载一个高电平信号。

(5) 引脚 $\overline{OE}$：输出允许控制信号。当对芯片进行读操作时，还必须将此信号置为有效。

(6) 引脚 V_{CC} 和 GND：分别为芯片的工作电源和接地线。

对图 4-13(b)的 DRAM 芯片来说，各引脚的功能如下。

(1) 引脚 $A_0 \sim A_{10}$：11 条地址信号线。很多 DRAM 芯片地址引脚数往往是实际所需地址信号线的一半，本芯片就属于这种情况。本来 4M 的字单元需要 22 条地址线($2^{22}=4M$)，但该芯片只有 11 条地址引脚。当 CPU 对该芯片进行读/写操作时，其地址经外围电路分为高 11 位行地址和低 11 位列地址，分先后送到芯片的地址端，并分别锁存到芯片内部的行、列地址锁存器中。

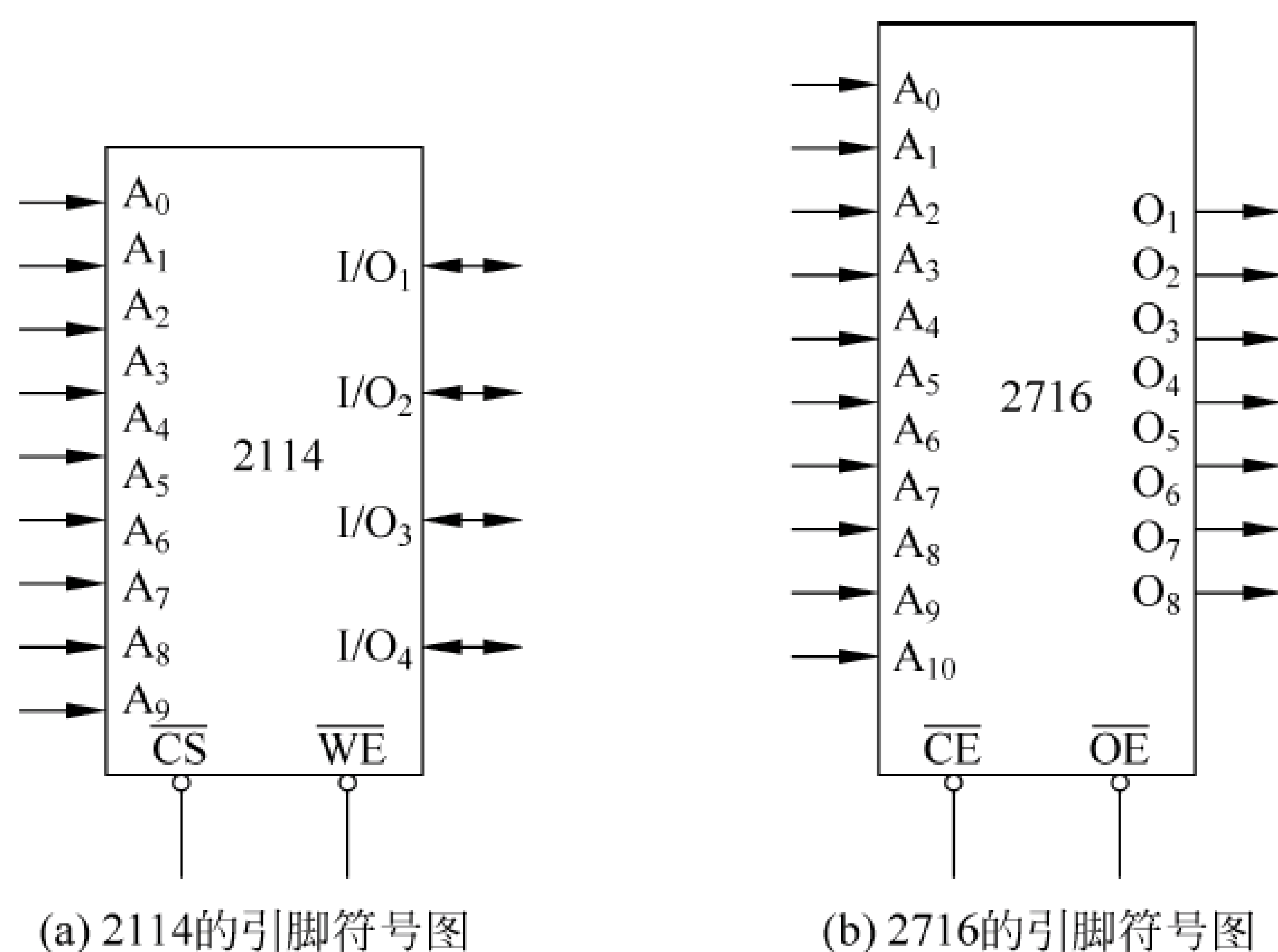
(2) 引脚 $D_1 \sim D_4$：4 条数据信号线，在标注上与上一芯片稍有不同。

(3) 引脚 $\overline{RAS}$、$\overline{CAS}$：分别用于行、列地址的锁存控制。

(4) 引脚 $\overline{OE}$ 和 $\overline{WE}$：同上。

图 4-13(c)的 EPROM 芯片的引脚与上述两个芯片的基本类似，只是多出了一个电源 V_{pp}，该电源引脚主要用于对芯片进行编程改写时加载一个 25V 的电压，使之在特殊条件下进行改写操作。

在有些场合，为突出芯片的引脚功能，也常常给出它们的引脚符号图。例如，前面讲到的静态 RAM 芯片 2114 和动态 RAM 芯片 2716 的引脚符号图如图 4-14 所示。



(a) 2114的引脚符号图　(b) 2716的引脚符号图

图 4-14　芯片引脚符号图

引脚符号图是将地址信号线、数据信号线和控制信号线分别列在芯片框的不同侧，并标明这些信号的输入输出方向，对于控制信号还需标明它们是低电平有效还是高电平有效。这种图可使人们对芯片的功能一目了然。

知识拓展

芯片的封装技术

人们经常听说某某芯片采用什么样的封装方式，在计算机中存在着各种各样不同的集成电路芯片，那么，它们又是采用何种封装形式的呢？下面对一些常见的芯片封装技术作一个介绍。

1）DIP

DIP(Dual In-line Package)是指采用双列直插形式封装的集成电路芯片，绝大多数中小规模集成电路芯片均采用这种封装形式，其引脚数一般不超过100个。采用DIP封装的CPU芯片有两排引脚，需要插入具有DIP结构的芯片插座上。当然，也可以直接插在有相同焊孔数和几何排列的电路板上进行焊接。DIP封装的芯片在从芯片插座上插拔时应特别小心，以免损坏引脚。Intel 8088 CPU和早期的内存芯片就采用这种封装形式。

2）QFP

QFP(Plastic Quad Flat Package)封装的芯片引脚之间距离很小，引脚很细，一般大规模或超大规模集成电路都采用这种封装形式，其引脚数一般在100个以上。用这种形式封装的芯片必须采用SMD(表面安装设备技术)将芯片与主板焊接起来。采用SMD安装的芯片不必在主板上打孔，一般在主板表面上有设计好的相应引脚的焊点。将芯片各脚对准相应的焊点，即可实现与主板的焊接。用这种方法焊上去的芯片，如果不用专用工具是很难拆卸下来的。Intel系列CPU中80286、80386和某些486主板采用这种封装形式。

3）PGA

PGA(Pin Grid Array Package)芯片封装形式在芯片的内外有多个方阵形的插针，每个方阵形插针沿芯片的四周间隔一定距离排列。根据引脚数目的多少，可以围成2～5圈。安装时，将芯片插入专门的PGA插座。为使CPU能够更方便地安装和拆卸，从486芯片开始，出现一种名为ZIF的CPU插座，专门用来满足PGA封装的CPU在安装和拆卸上的要求。ZIF(Zero Insertion Force)是指零插拔力的插座，把这种插座上的扳手轻轻抬起，CPU就可以很容易、轻松地插入插座中，然后将扳手压回原处，利用插座本身的特殊结构生成的挤压力，将CPU的引脚与插座牢牢地接触，绝对不存在接触不良的问题。而拆卸CPU芯片只需将插座的扳手轻轻抬起，则压力解除，CPU芯片即可轻松取出。Intel系列CPU中，80486和Pentium、Pentium Pro均采用这种封装形式。

4）BGA

随着集成电路技术的发展，对集成电路的封装要求更加严格，当IC的引脚数大于208 Pin时，传统的封装方式比较困难。因此，除使用QFP等封装方式外，现今大多数的高脚数芯片(如图形芯片与芯片组等)皆转而使用BGA(Ball Grid Array)封装技术。BGA一出现便成为CPU、主板上南/北桥芯片等高密度、高性能、多引脚封装的最佳选择。Intel公司在其CPU Pentium Ⅱ、Pentium Ⅲ、Pentium 4等，以及芯片组中开始使用BGA。

5）CSP

随着全球电子产品个性化、轻巧化的需求逐渐增加，封装技术已进步到 CSP（Chip Size Package）。它减小了芯片封装外形的尺寸，做到裸芯片尺寸有多大，封装尺寸就有多大。即封装后的 IC 尺寸边长不大于芯片的 1.2 倍。

CSP 封装适用于脚数少的 IC，如内存条和便携电子产品。未来则将大量应用在信息家电（IA）、数字电视（DTV）、电子书（E-Book）、无线网络（WLAN）、Gigabit Ethemet、ADSL、手机芯片、蓝牙（Bluetooth）等新兴产品中。

2. 半导体存储器读写周期

前面已经提到，在选择存储器芯片构成计算机主存储器时，要考虑 CPU 与存储器之间的速度匹配。存储器在出厂时，其存取速度就已经确定，厂家在对芯片的有关技术说明中会给出其存取速度的相关技术参数，这一技术参数主要是通过存储器的读写周期（又称为存储周期）来反映的，而读写周期是通过波形图（又称时序图）来体现的。下面以静态 RAM 芯片 2114 为例对存储器的读写周期加以说明。

2114 的读周期波形图如图 4-15 所示。

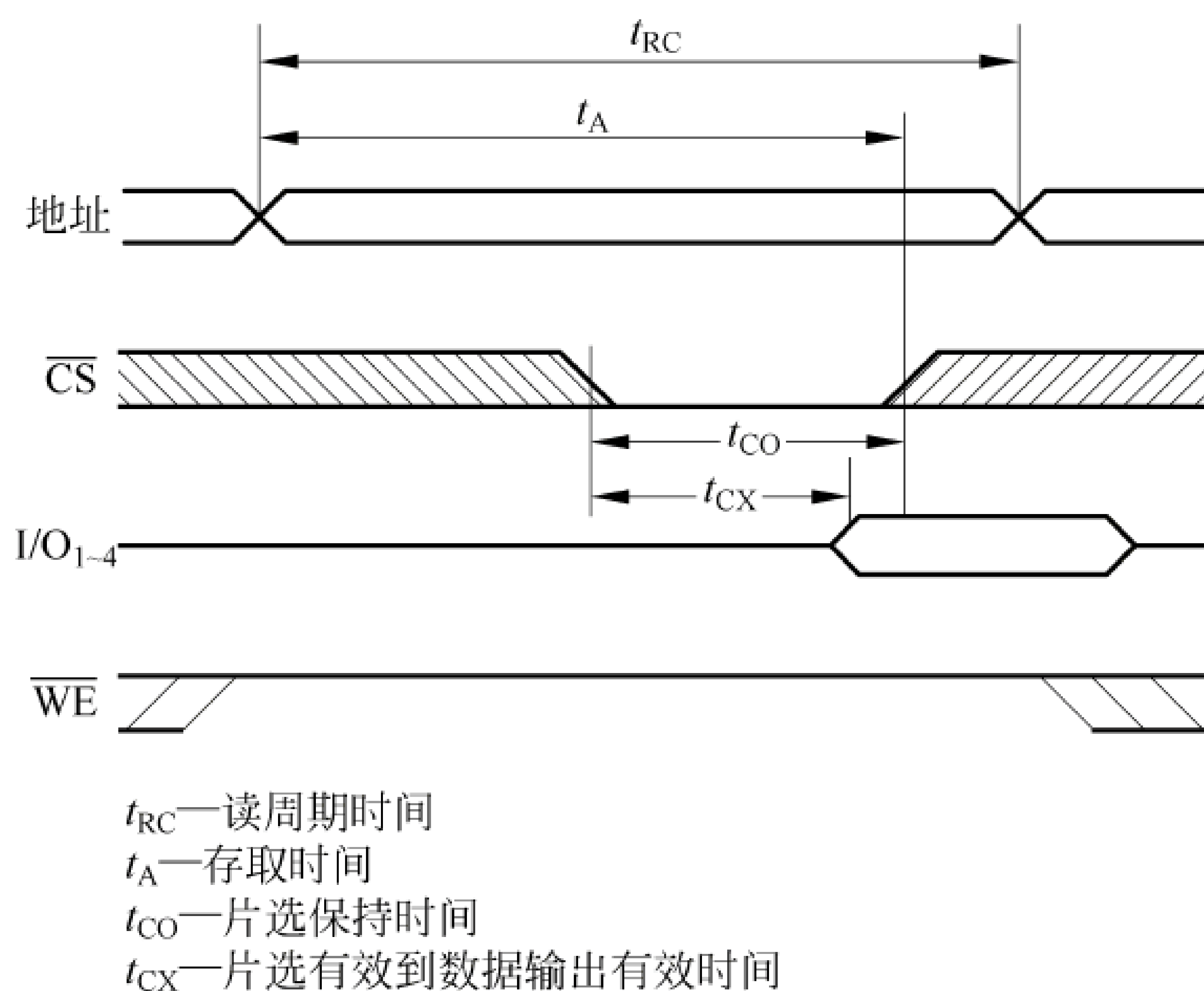


图 4-15 2114 的读周期波形

其中，读周期时间 t_{RC} 就是指 CPU 对 2114 芯片进行读操作所需要的时间，这一时间一方面说明，若要从 2114 中读取一个字单元，至少需要经过 t_{RC} 的时间；另一方面说明，若要完成对 2114 的正确读操作，CPU 加载在 2114 芯片地址引脚上的地址信号需要至少维持 t_{RC} 的时间。

时间 t_A 表示 2114 在 CPU 加载的地址有效后，经过 t_A 时间后可以将数据读出到数据线上。另外，为使芯片工作，片选信号 $\overline{CS}$ 还必须紧跟地址信号之后产生，且应在数据有效读出的 t_{CX} 时间之前给出，且必须维持至少 t_{CO} 的时间。

CPU 是通过一个读周期来完成对存储器的读操作的，而读周期产生的地址、控制等信号只有满足以上时序要求，才能保证读操作的正确性。因此，在进行机器硬件系统设计时，一方面要保证逻辑电路设计的正确性，另一方面还要保证时序设计的正确性，两者缺一

不可。

2114 的写周期波形图如图 4-16 所示。

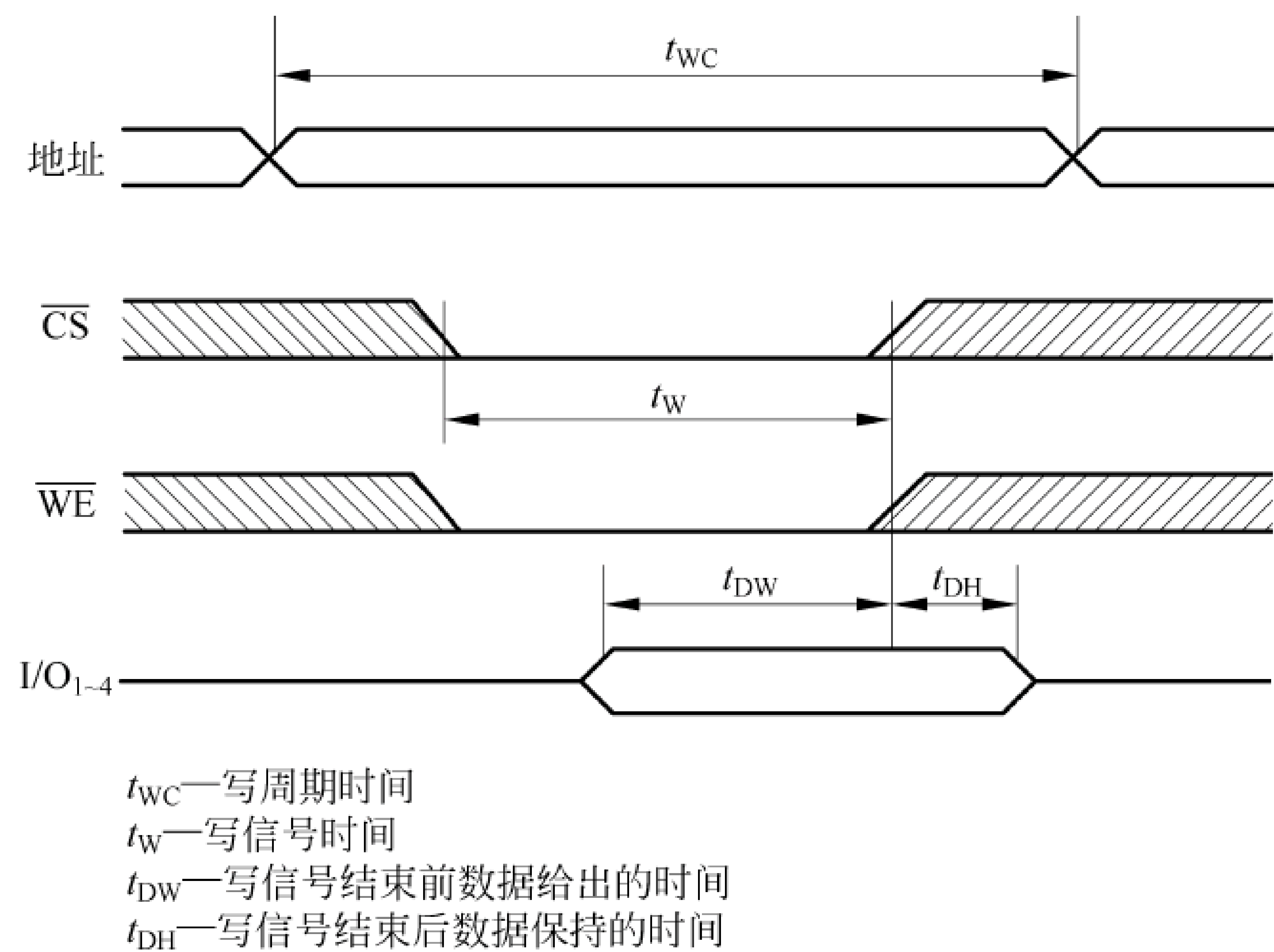


图 4-16　2114 的写周期波形

其中,写周期时间 t_{WC} 就是指 CPU 对 2114 芯片进行写操作所需要的时间,也就是说,若要向 2114 中写入一个字单元,需要至少 t_{WC} 的时间。一般来讲,半导体存储器的读周期时间和写周期时间是相同的,这一时间就是存储器芯片的存取时间。存取时间越小,存储器的速度就越快;反之,存取时间越大,存储器的速度也就越慢。

动态存储器和只读存储器的读写周期与静态存储器的类似,此不再赘述。

3. 半导体存储器与 CPU 的连接

在构成计算机主存储器时,还需要根据机器容量的要求和所选用的半导体存储器芯片容量的情况进行综合设计。当单片存储器芯片的容量不能直接满足主存储器容量的要求时,需要选择多片进行容量扩展连接,以构成主存储器模块。下面分三种情况介绍存储器的容量扩展连接。

1) 位扩展连接

设主存储器的容量为 $M\times N$ 位,而选用的存储器芯片的容量为 $M\times n$ 位,其中,N 是 n 的整数倍。在这种情况下,单片存储器芯片的字单元数与主存储器的相同,但每个字单元的位数不能满足主存储器的要求。这时,需要进行一种位扩展连接,使用 N/n 片芯片并联起来,如图 4-17 所示,具体的连接方法是:

(1) 所有芯片的地址线 A 对应连接在一起。

(2) 所有芯片的片选信号线 $\overline{CE}$ 对应连接在一起。

(3) 所有芯片的读写 $\overline{WE}$ 线对应连接在一起。

(4) 每个芯片的数据线各自单独引出。

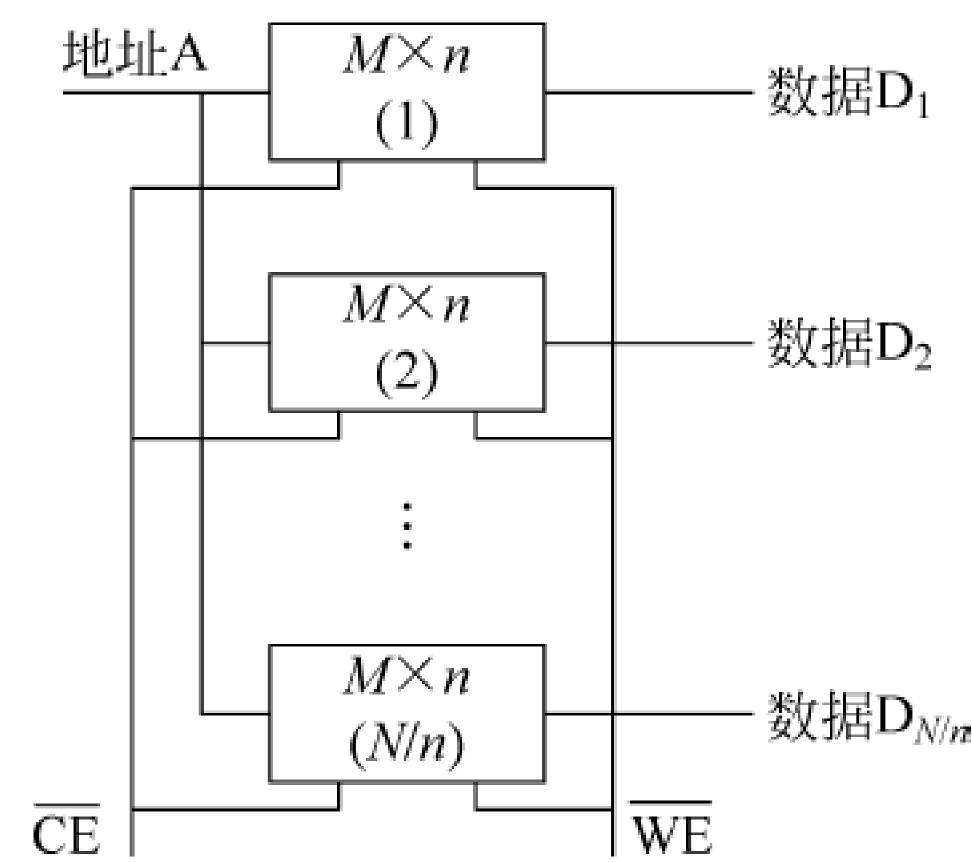


图 4-17　位扩展连接

采用这种方式连接后，每一次 CPU 来的地址同时选中所有芯片的同一个字单元，且地址线数未增加，也就意味着整个存储体的字数为 M；$\overline{CE}$同时选中所有芯片工作，而$\overline{WE}$同时使所有芯片读或写；最后，由于各芯片的数据线是单独引出的，被同时选中的芯片各自有 n 条数据线引出，加起来正好 N 条，若将这 N/n 个芯片看成一个整体，那么其容量就是 $M \times N$ 位，正好符合主存储器的要求。

下面举一个例子进一步说明位扩展连接的存储器芯片与 CPU 的连接。

【例 4.1】 使用一种 64M×4b 的存储器芯片构成 64M×16b 的主存储器，并与一个 16b 的 CPU 连接。

解：64M×4b 的存储器芯片构成 64M×16b 的主存储器需要的芯片数为 16/4＝4(片)，并进行位扩展连接，如图 4-18 所示。

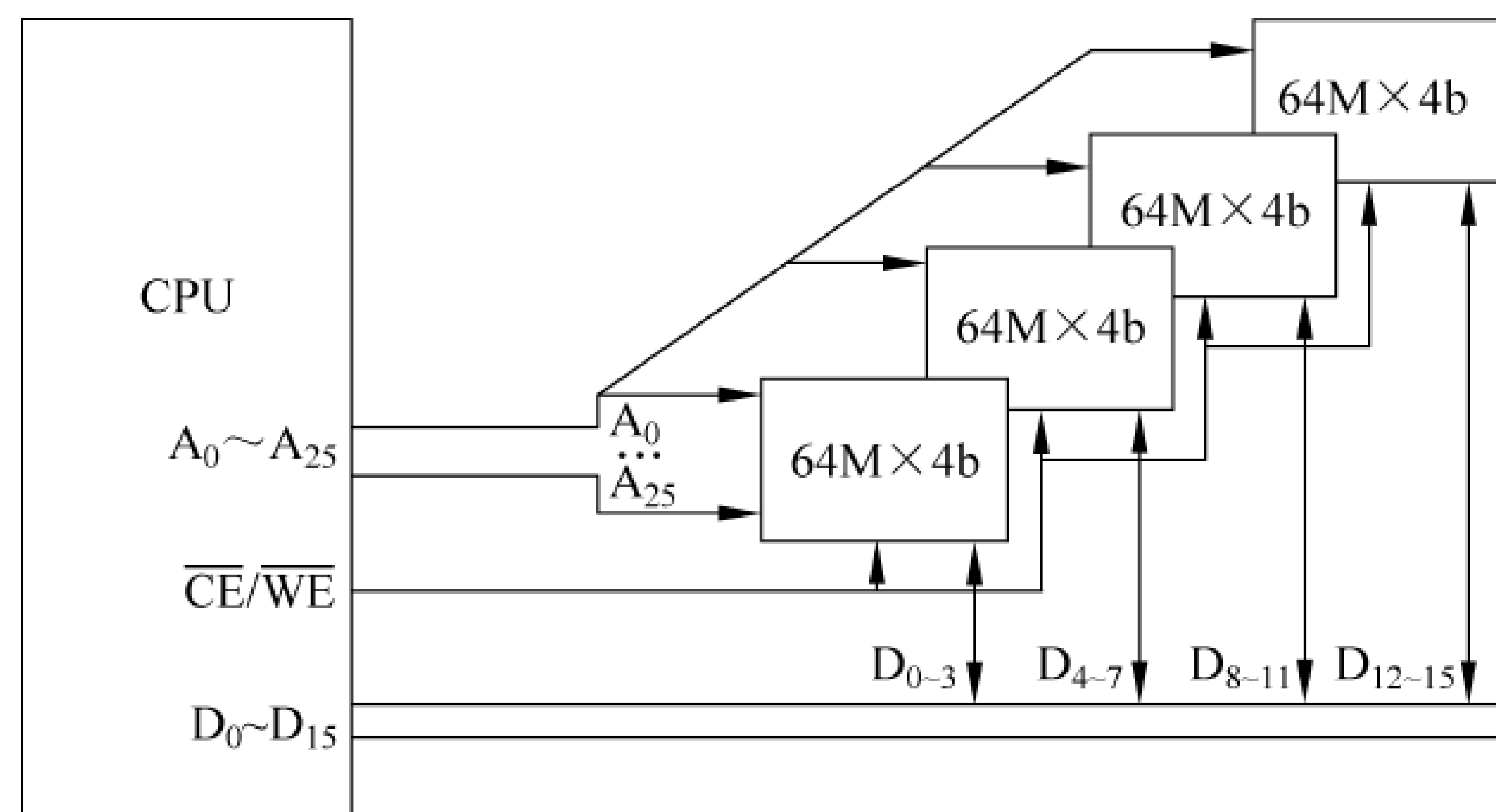


图 4-18 芯片位扩展与 CPU 的连接

2）字扩展连接

字扩展连接的情况是：设主存储器的容量为 $M \times N$ 位，而选用的存储器芯片的容量为 $m \times N$ 位，其中，M 是 m 的整数倍。在这种情况下，单片存储器芯片的字单元位数与主存储器的相同，但每个芯片的字单元数不能满足主存储器的要求。这时，需要进行一种字扩展连接，使用 M/m 片芯片连接起来，如图 4-19 所示，具体连接方法是：

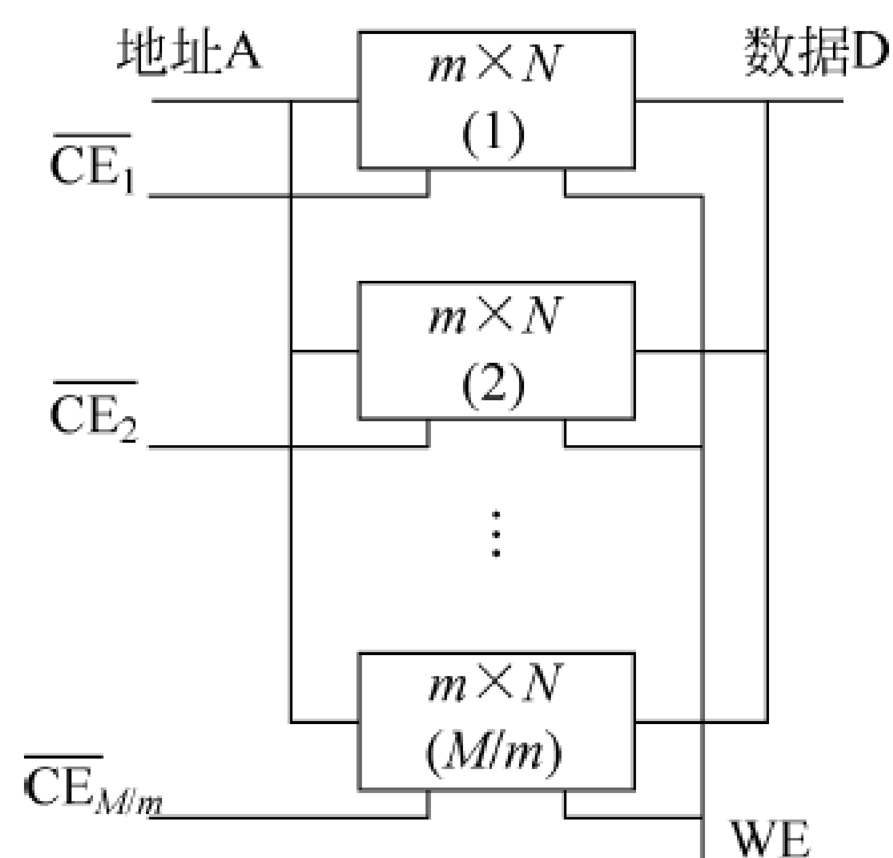


图 4-19 字扩展连接

（1）所有芯片的地址线 A 与 CPU 的低位对应地址线连接在一起。

（2）所有芯片的数据线对应连接在一起。

（3）所有芯片的读写$\overline{WE}$线对应连接在一起。

（4）每个芯片的片选信号线$\overline{CE}$各自单独引出，并由 CPU 剩余的部分高位地址线产生。

采用这种方式连接后，由于所有芯片的数据线是并联在一起的，所以数据位数不变，每次读写的为 N 位；CPU 来的低位地址同时选中所有芯片的同一个字单元，再由高位译码产生的片选信号$\overline{CE}$一次选中一个芯片工作，$\overline{WE}$控制被选中的芯片读或写，共有 M/m 个容量为 $m \times N$ 位的芯片，总容量就是 $M \times N$ 位，正好符合主存储器的要求。

下面举一个例子进一步说明字扩展连接的存储器芯片与 CPU 的连接。

【例 4.2】 使用一种 16M×16b 的存储器芯片构成 64M×16b 的主存储器，并与一个 16b 的 CPU 连接。

解：16M×16b 的存储器芯片构成 64M×16b 的主存储器需要的芯片数为 16/4＝4(片)，并进行字扩展连接，如图 4-20 所示。

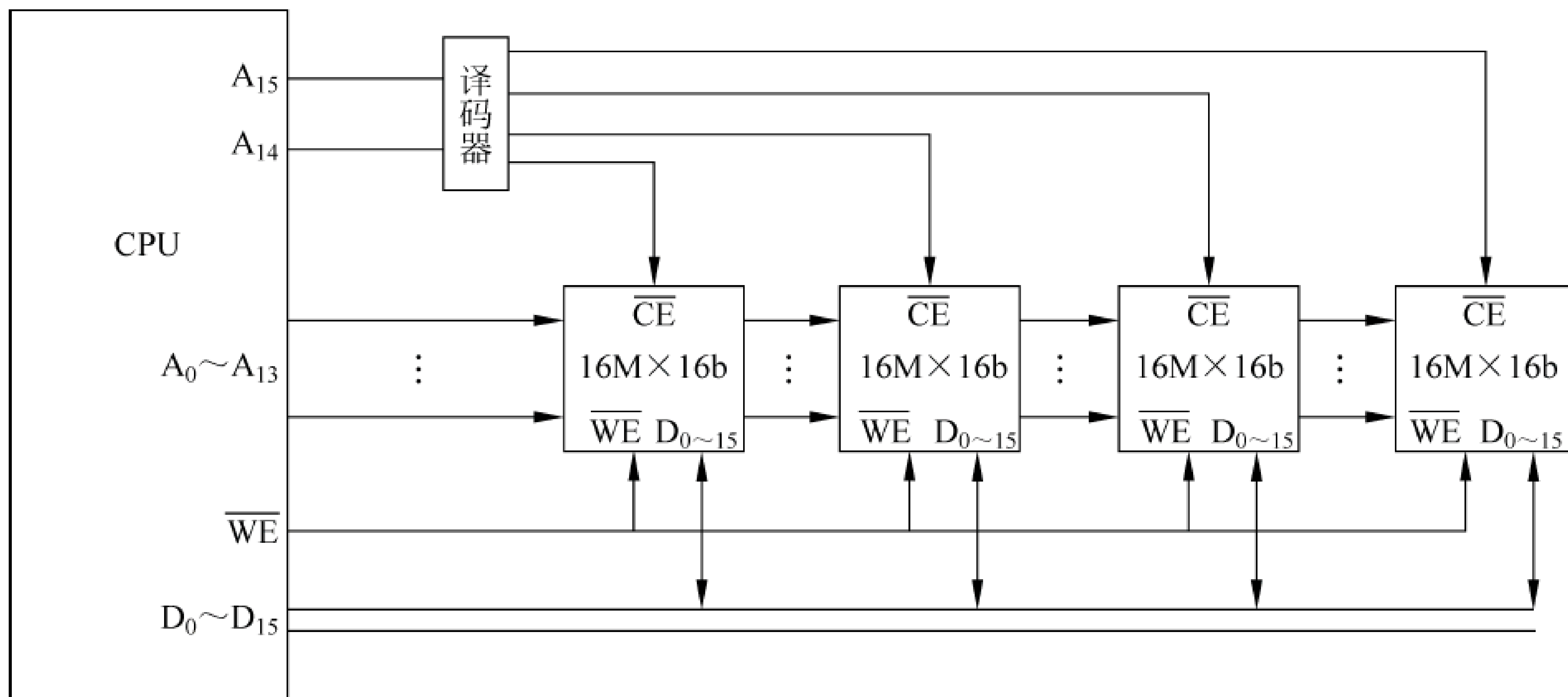


图 4-20 芯片字扩展与 CPU 的连接

其中，CPU 的地址线 A_0～A_{13} 与每个芯片的对应地址线连接，A_{14} 和 A_{15} 则用于译码，产生对 4 个芯片的片选控制信号，且这 4 个片选信号同时只有一个为有效，选中一个芯片工作。CPU 的数据线 D_0～D_{15} 与每个芯片的对应数据线连接。

3）字位扩展连接

字位扩展连接的情况是：设主存储器的容量为 $M\times N$ 位，而选用的存储器芯片的容量为 $m\times n$ b，其中，M、N 分别是 m、n 的整数倍。在这种情况下，单片存储器芯片的字单元数和字单元的位数均不能满足主存储器的要求。这时，需要进行一种字位扩展连接，使用$(M/m)\times(N/n)$个芯片连接起来，连接方法就是将以上两种情况结合起来考虑。

4.3 交叉存储技术

1. 交叉存储技术的基本思想

先来看一个生活中的例子。假设某一产品需要两道生产工序完成，分别是工序 1 和工序 2，且两道工序采用流水作业的方式进行。现在安排两个工人 A、B 分别做工序 1 和工序 2，其中工人 A 完成工序 1 的时间是 Δt，而工人 B 完成工序 2 的时间是 $3\Delta t$。那么，在这种情况下，流水线将每隔 $3\Delta t$ 的时间生产出一个该产品。其中，工人 B 需连续不间断地工作，而工人 A 则可以每工作 1 个 Δt 时间休息 2 个 Δt 的时间，如图 4-21 所示。显然，在这种流水作业的情况下，工人 A 未满负荷工作，流水线的效率也未能真正发挥出来。

现在改变一下流水模式，增加 2 个工作速度与 B 相同的工人 C、D 都来做工序 2，这样就是工人 A 一人做工序 1，而工人 B、C、D 三人同时做工序 2。在 4 个工人都满负荷工作的情况下，流水线将每隔 Δt 的时间生产出一个该产品，如图 4-22 所示。

图 4-21 流水线生产模式 1

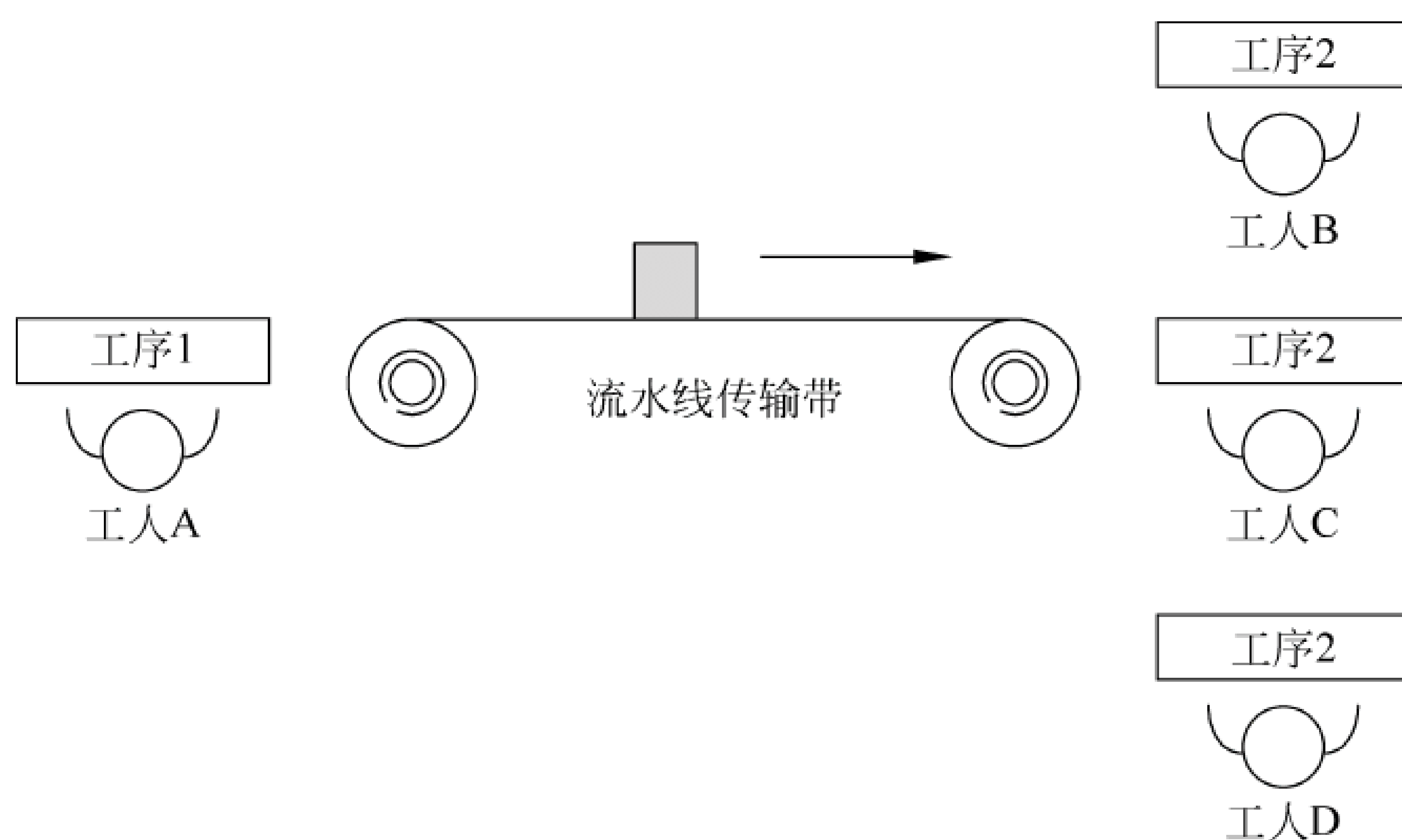


图 4-22 流水线生产模式 2

显然，后一种模式比前一种模式生产效率提高了两倍，而其代价只是增加了两个工人，不是增加 4 个工人和两套流水线设备。

可以吸取这一思想，应用到计算机 CPU 与存储系统的配置上。由于 CPU 的速度远比存储器的速度快，当采用单 CPU 和单存储器模块时，CPU 每进行一次访存操作，都需要等待存储器的操作，且一个访存周期只能从单个存储器模块中存取一个字单元。如果多配置几个存储器模块，让 CPU 与多模块的存储器之间采用后一种流水模式进行工作，那么，就可以实现一个访存周期能从多个存储器模块中存取多个字单元。

2. 交叉存储器模块的组织

为了实现 CPU 在一个访存周期能从多个存储器模块中存取多个字单元的目标，还需要对存储器模块的组织进行合理地配置。

传统的单模块存储器的单元地址是进行线性编址的，也就是从一个最低地址（通常是二进制全 0）开始连续编址到一个最高地址（通常是二进制全 1）。现在采用多模块存储器结构，也就相当于将传统的单模块存储器拆分为多个模块。那么，这多个模块如何编址是人们首先要研究的问题。

假设存储器共有 32 个字单元，分为 4 个模块，每个模块有 8 个字单元。考察以下两种编址方式：一是每个模块仍然像传统存储器一样进行线性编址，即模块 1 的单元地址是 0～7，模块 2 的单元地址是 8～15，模块 3 的单元地址是 16～23，模块 4 的单元地址是 24～31，如图 4-23(a)所示；二是采用交叉编址的方式，即模块间的单元地址是连续的，而模块内的单元地址是不连续的，如图 4-23(b)所示。

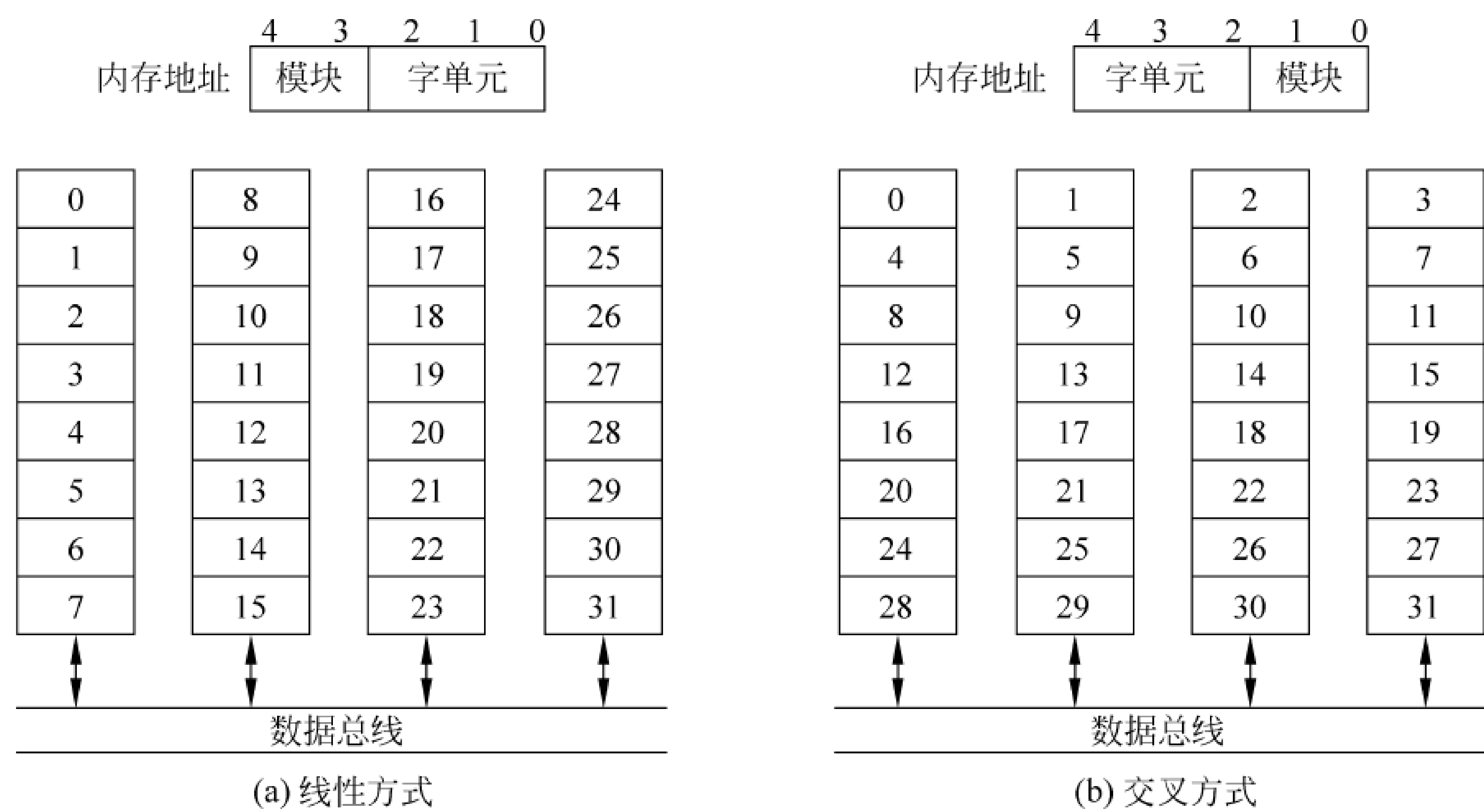


图 4-23　存储器模块的组织方式

在采用图 4-23(a)的线性方式编址的情况下，当 CPU 要同时访问多个连续的存储单元时，这些单元分布在同一模块的概率是最大的。例如，CPU 在某一访存周期需向存储器读 4 个字单元的数据，这 4 个字单元地址是连续的，那么它们只有可能分布在一个或两个存储模块中。由于对于每个存储模块来说，一个访存周期只能读写一个字单元，因此，采用这种线性编址方式并不能使 CPU 在一个访存周期存取多个字单元。

而采用图 4-23(b)的交叉编址方式则不同，当 CPU 要同时访问多个连续的存储单元时，由于交叉编址的特殊性，使得这些单元会分布在不同模块。例如，CPU 在某一访存周期需向存储器读 4 个字单元的数据，这 4 个字单元地址是连续的，那么它们会分别分布在 4 个存储模块中。由于每个存储模块均能独立地进行读写操作，这就使得 CPU 在一个访存周期能够同时存取多个字单元。

由此可以看出，当使用多模块存储器结构时，对模块的编址需采用交叉编址方式，才能真正发挥出多模块存储器的效率，真正实现 CPU 在一个访存周期同时存取多个字单元的目标。

不过要注意的是，在这种结构下，CPU 是通过同一条数据总线与多个存储模块之间进行数据传送的，而数据总线一次只能容纳一个字单元数据。因此，这多个存储模块必须采取在时间上交错开来的方式与 CPU 通过数据总线交换数据，CPU 在同一访存周期存取的多个字单元实际上并不是完全“同时”的，严格来讲是顺序存取的，但这并不影响所实现的 CPU 一个访存周期存取多个字单元的目标。当然，能实现这一目标的前提是 CPU 的总线操作速度远比存储器的读写速度快，理想情况下，若 CPU 的总线操作速度是存储器的读写速度的 n 倍，则应配置不少于 n 个存储模块与 CPU 相匹配。

下面再通过定量分析，比较一下传统的单模块和多模块交叉编址方式的存取速度。假设每个模块的字单元长度等于数据总线宽度，存储器模块的存储周期为 T，CPU 的总线传送周期为 τ(即 CPU 从总线读取一个字单元或向总线写一个字单元的时间为 τ)，存储器的交叉模块数为 m，为了实现流水线方式存取，应当满足

$$T = m\tau \tag{4.1}$$

则 CPU 与单模块存储器和采用交叉编址方式的多模块存储器之间的数据存取示意图分别如图 4-24(a)、图 4-24(b)所示。

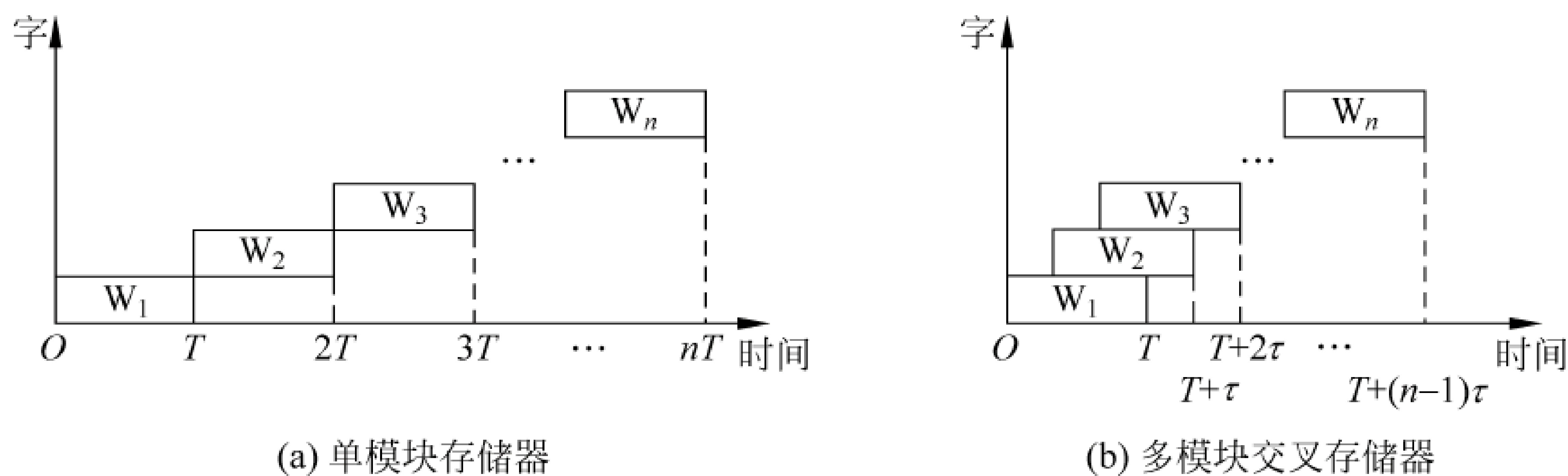


图 4-24 存储模块的存取方式

从图 4-24 中可以看出，CPU 向单模块存储器存取 n 个数据所需的总时间为 nT，采用的是完全顺序存取方式；而 CPU 向多模块交叉存储器存取 n 个数据所需的总时间为 $T+(n-1)\tau$，采用的是在时间上交错开来的并行存取方式。把这连续的 n 个字单元数据的存取看成一个流水操作，则前一种方式是每间隔 T 时间存取一个数据，而后一种方式是每间隔 τ 的时间存取一个数据，CPU 的存取速度得到了大大提高。

3. 交叉存储器的组成

为实现多模块交叉存储器的功能，光对这多个模块进行交叉编址是不够的，还需对 CPU 访存的读写电路进行设计。在多模块交叉存储系统中，为保证每个模块均能够进行独立的读写操作，需为每个存储模块配置独立的读写电路，并由一个存储器控制部件进行顺序控制。

如图 4-25 所示是一个四模块交叉存储器的组成结构图。图中，4 个存储模块各配置了一套独立的读写电路，它们在一个存储器控制部件的统一控制下，分时地使用数据总线完成与 CPU 之间的数据传送。

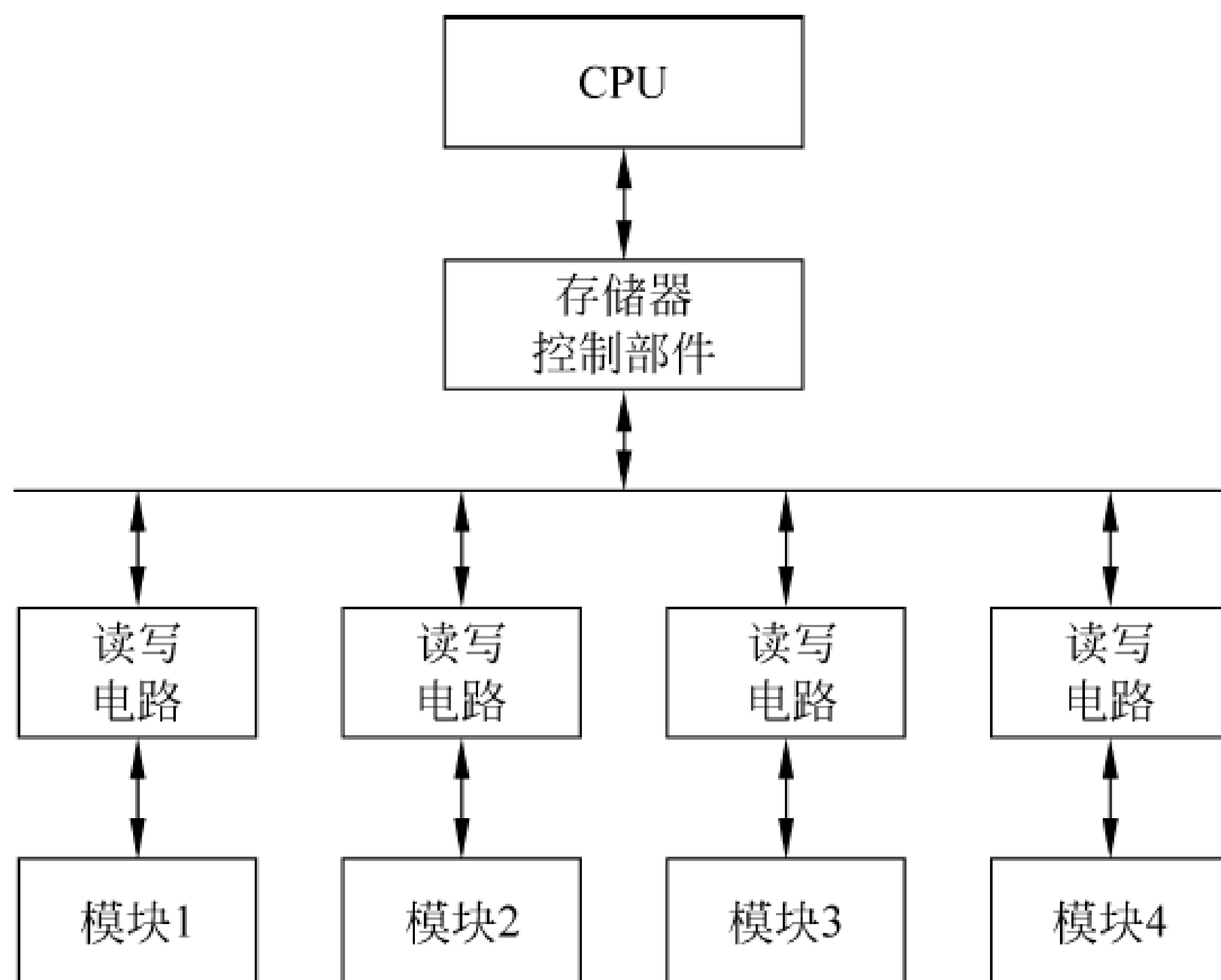


图 4-25 多模块交叉存储器的组成

4.4　高速缓冲存储器

半导体存储器在制作工艺上有所不同，不同制作工艺的存储器在存取速度上也有所差异。一般来讲，速度越快的半导体存储器其价格也越高。高速缓冲存储器技术是一种使用小容量快速半导体存储器件实现提高 CPU 访存速度的有效方法，也是现代计算机普遍采用的一种方法。

4.4.1　cache 实现的基本原理

1. 程序的局部性原理

对大量典型程序运行情况的分析结果表明，在一个较短的时间间隔内，CPU 对一个程序指令的访问往往集中在一个较小的存储器逻辑地址空间范围内，也就是说，CPU 取指令和取数据的操作具有时间上局部分布的倾向，这种现象称为程序访问的局部性。

其实，程序访问的局部性与所编制程序的结构特点是紧密相关的。人们都知道，程序从控制结构上讲主要包括顺序结构、分支结构、循环结构和主子程序结构等。一般来讲，一个典型的程序中大量使用的是顺序结构，也就是说，程序大部分情况下是顺序执行的，发生转移和过程调用的情况相对较少。尤其是现代程序的编制强调程序的模块化，注重程序的规整性和结构化。例如，从 20 世纪 70 年代开始出现的 PASCAL 语言就是一个典型的模块化语言，它强烈建议程序员尽可能少地使用 goto 语句。再如，C 语言是一个函数化语言，它的基本组成就是函数，而函数可以看成一个个程序模块。当 CPU 在某一段时间内执行程序的一个小的模块时，CPU 对程序的访问往往就局限在这一模块所分布的局部存储空间内，加之程序中大量使用循环，使程序的执行更有局部性倾向。虽然程序中不可避免要使用分支转移，但一方面这种转移情况相对较少，另一方面，即使发生了转移，程序又将进入另一模块进行局部性访问。

当一个程序执行时，是由操作系统将其从计算机辅存中调入主存中的。操作系统在为程序分配主存空间时，是尽可能为程序分配连续的存储空间。这样就使得程序在存储器中的存放顺序与其编写顺序相一致，这也就进一步保证了 CPU 对程序访问的局部性。

2. cache 的基本思想和工作原理

在前面 4.1.3 节介绍存储系统的层次结构时讲到，现代存储器为了满足用户对存储器高速度、大容量和低价格的要求，普遍采用了 cache-主存-辅存这样一种结构。实际上，这里面包括了两种实现不同目标的层次结构：一是“cache-主存”结构，使用高速缓冲存储器提高 CPU 的访存速度；二是“主存-辅存”结构，使用虚拟存储器管理为用户提供一个大容量的程序空间。

实现 cache 的基本思想基于的就是程序的局部性原理：在 CPU 与主存之间设置一个小容量的高速缓冲存储器 cache，当一个程序调入主存运行时，将该程序当前要执行的指令及其后将执行的一部分指令同时调入 cache 中，CPU 每次首先从 cache 中取指令执行，根据程序的局部性原理，CPU 大部分情况下可以在 cache 中取到指令(称为命中)，只要命中率

足够高，就可以使得 CPU 访存的速度接近于访 cache 的速度。

下面通过图 4-26 进一步阐述 cache 的工作原理。

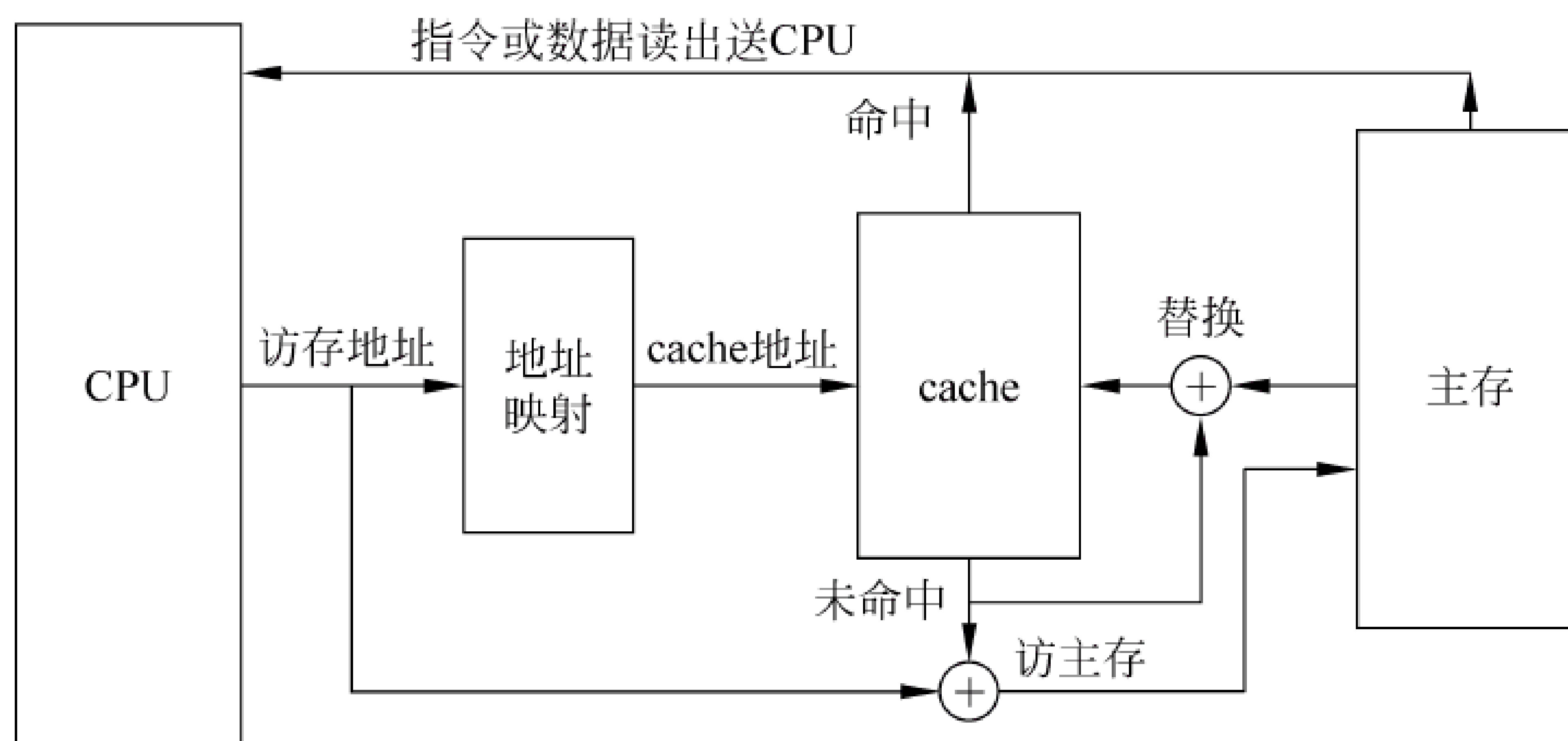


图 4-26 cache 工作原理图

CPU 的访存地址（主存地址）首先通过一个主存-cache 的地址映射机构转换成一个访 cache 的地址。如果 CPU 要访问的这一单元内容已经在 cache 中，则本次访 cache 命中，于是从 cache 中将该单元的内容读出，并送 CPU；若 CPU 要访问的这一单元不在 cache 中，则本次访 cache 不命中，于是使用 CPU 的访存地址直接访问主存，从主存中将该单元的内容读出，并送 CPU，从而完成了一次 CPU 的访存操作。在 cache 存储器组织中，cache 和主存都按块进行组织，每一个块由若干个字单元组成，且主存的块和 cache 的块大小相同。在不命中的情况下，CPU 访存结束后，还需将本次所访问的单元所在主存的块的全部单元内容调入 cache 中。若当前 cache 已满，则需将 cache 中的一个旧块替换出来。

对以上 cache 工作原理图进行进一步分析会发现，CPU 无论是命中而访 cache 还是未命中而访主存，都需要首先使用访主存地址进行一次地址映射操作，地址映射的目的是将 CPU 来的访主存地址转换成一个访 cache 地址，同时通过地址映射能够判断本次要访问的主存地址字单元是否已经在 cache 中。这就带来了一个问题，那就是 CPU 无论是访 cache 还是访主存之前不是又额外增加了一个操作吗？这增加的额外操作会不会增加 CPU 访存的时间呢？确实，地址映射是实现 cache 额外增加的操作，但是，为了不影响 CPU 的访存速度，地址映射是通过使用一个称为相联存储器的高速器件完成的。

前面所介绍的存储器（半导体 RAM 或 ROM 等）都是按地址访问的存储器，也就是说，CPU 通过给出一个存储单元的地址对选中的存储单元的内容进行读写操作。而相联存储器则不同，它是按内容访问的存储器。具体来说就是，把存储单元中的内容的一部分或全部作为关键字，去检索整个存储器，然后对与检索关键字相符的存储单元进行读写操作。

相联存储器的组成原理图如图 4-27 所示。它主要由数据寄存器、屏蔽寄存器、输出寄存器、匹配寄存器、存储体和比较逻辑等组成。

数据寄存器用来存放要按内容检索的关键字，其位数和相联存储器的存储单元位数相等。

屏蔽寄存器用来存放屏蔽码，屏蔽码用于控制数据寄存器中哪些位参与检索比较（0 屏蔽，1 开放），其位数和数据寄存器位数相同。

输出寄存器用来存放按数据寄存器中的关键字检索所得到的比较相符的单元内容，其

位数和数据寄存器位数相同。

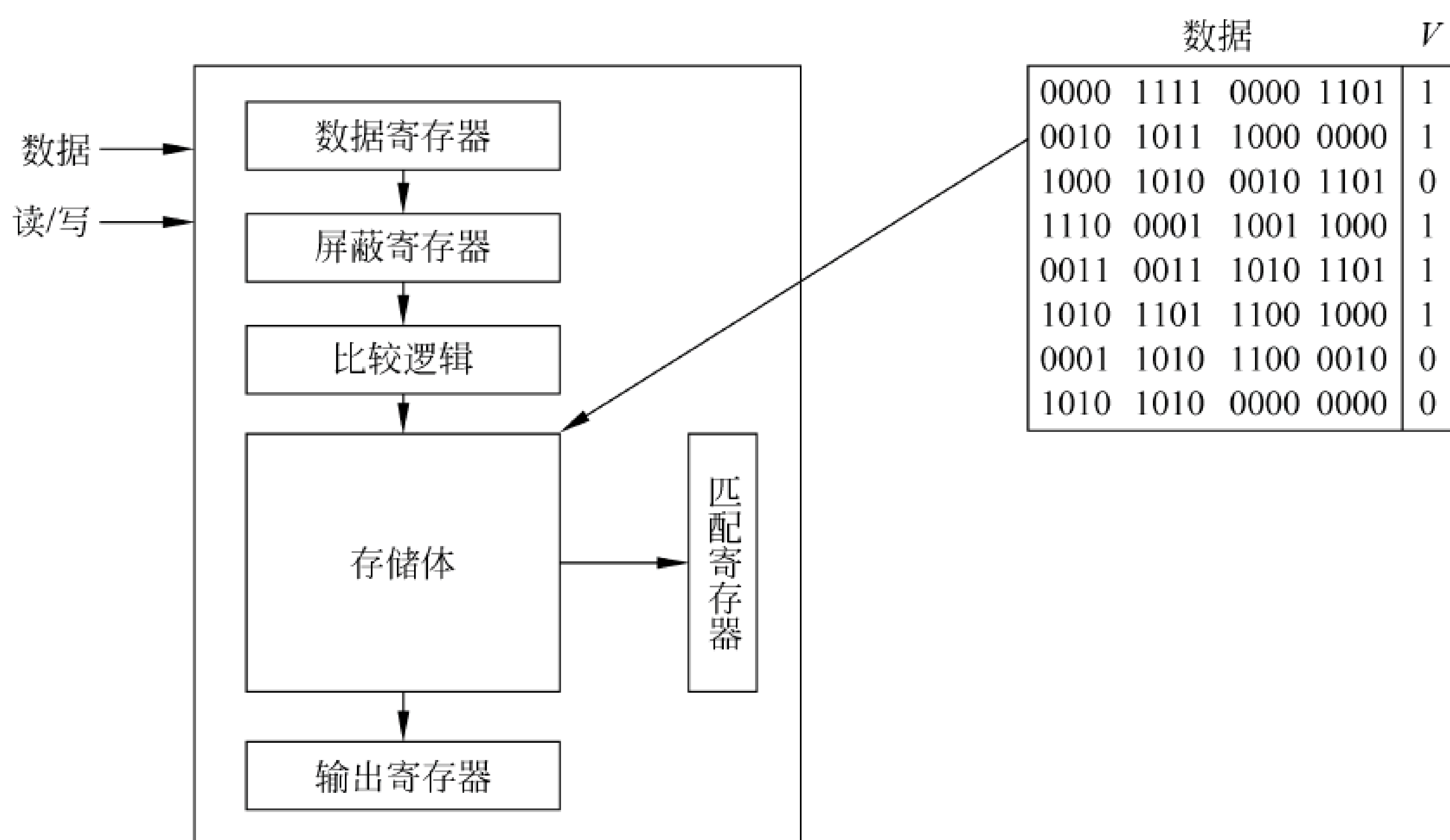


图 4-27 相联存储器组成原理图

匹配寄存器用来存放按数据寄存器中的关键字检索比较相符的单元标志，其位数等于相联存储器的存储单元数，比较相符的单元在匹配寄存器的相应位被标识为 1。

为了说明问题，假设一个相联存储器的存储体由 8 个字单元构成，每个字单元为 16 位。存储体的每个字单元除了存储一个 16 位字外，还设有一个标志位 V，用于指明该单元的数据是否有效，1 为有效，0 为无效。

当 CPU 访问相联存储器时，首先将一个要按内容检索的关键字送相联存储器的数据寄存器，然后再送一个屏蔽字到屏蔽寄存器。例如，假设 CPU 要检索在存储器中是否有高 4 位为 1010 的单元，则 CPU 将一个关键字 1010 xxxx xxxx xxxx 送相联存储器的数据寄存器，然后再送一个屏蔽字 1111 0000 0000 0000 到屏蔽寄存器。于是，相联存储器开始将所有标志位 V 为 1 的单元的高 4 位同时与数据寄存器中的 1010 进行比较，若有相符者，则将匹配寄存器与比较相符的单元相对应的位置 1，并同时把相符的单元内容读出到输出寄存器中。至此，CPU 的一次访相联存储器操作结束。

值得注意的是，相联存储器的比较是所有单元同时进行的，而且是全硬件实现，因此，这种操作速度快。对于 cache 实现来说，通过相联存储器完成地址映射，额外增加的时间是非常少的。

4.4.2 主存与 cache 的地址映射

CPU 对存储器的访问，通常是一次读写一个字单元。当 CPU 访 cache 不命中时，需将存储在主存中的字单元连同其后若干个字一同调入 cache 中，之所以这样做，是为了使其后的访存能在 cache 中命中。因此，主存和 cache 之间一次交换的数据单位应该是一个数据块。数据块的大小是固定的，由若干个字组成，且主存和 cache 的数据块大小是相同的。

从 cache-主存层次实现的目标看，一方面既要使 CPU 的访存速度接近于访 cache 的速度，另一方面为用户程序提供的运行空间应保持为主存容量大小的存储空间。在采用

cache-主存层次的系统中，cache 对用户程序而言是透明的，也就是说，用户程序可以不需要知道 cache 的存在。因此，CPU 每次访存时，依然和未使用 cache 的情况一样，给出的是一个主存地址。但在 cache-主存层次中，CPU 首先访问的是 cache，并不是主存。为此，需要一种机制将 CPU 的访主存地址转换成访 cache 地址。而主存地址与 cache 地址之间的转换是与主存块及 cache 块之间的映射关系紧密联系的，也就是说，当 CPU 访 cache 未命中时，需要将欲访问的字所在主存中的块调入 cache 中，按什么样的策略调入，直接影响到主存地址与 cache 地址的对应关系，这也就是本小节要解决的主存与 cache 的地址映射问题。

主要有三种地址映射方式，分别为全相联映射、直接相联映射和组相联映射。

1. 全相联映射

全相联映射是指主存中任意一块都可以映射到 cache 中任意一块的方式，也就是说，当主存中的一块需调入 cache 时，可根据当时 cache 的块占用或分配情况，选择一个块给主存块存储，所选的 cache 块可以是 cache 中的任意一块。例如，设 cache 共有 2^C 块，主存共有 2^M 块，当主存的某一块 j 需调进 cache 中时，它可以存入 cache 的块 0、块 1、…、块 i、…、块 2^C-1 的任意一块上，如图 4-28 所示。

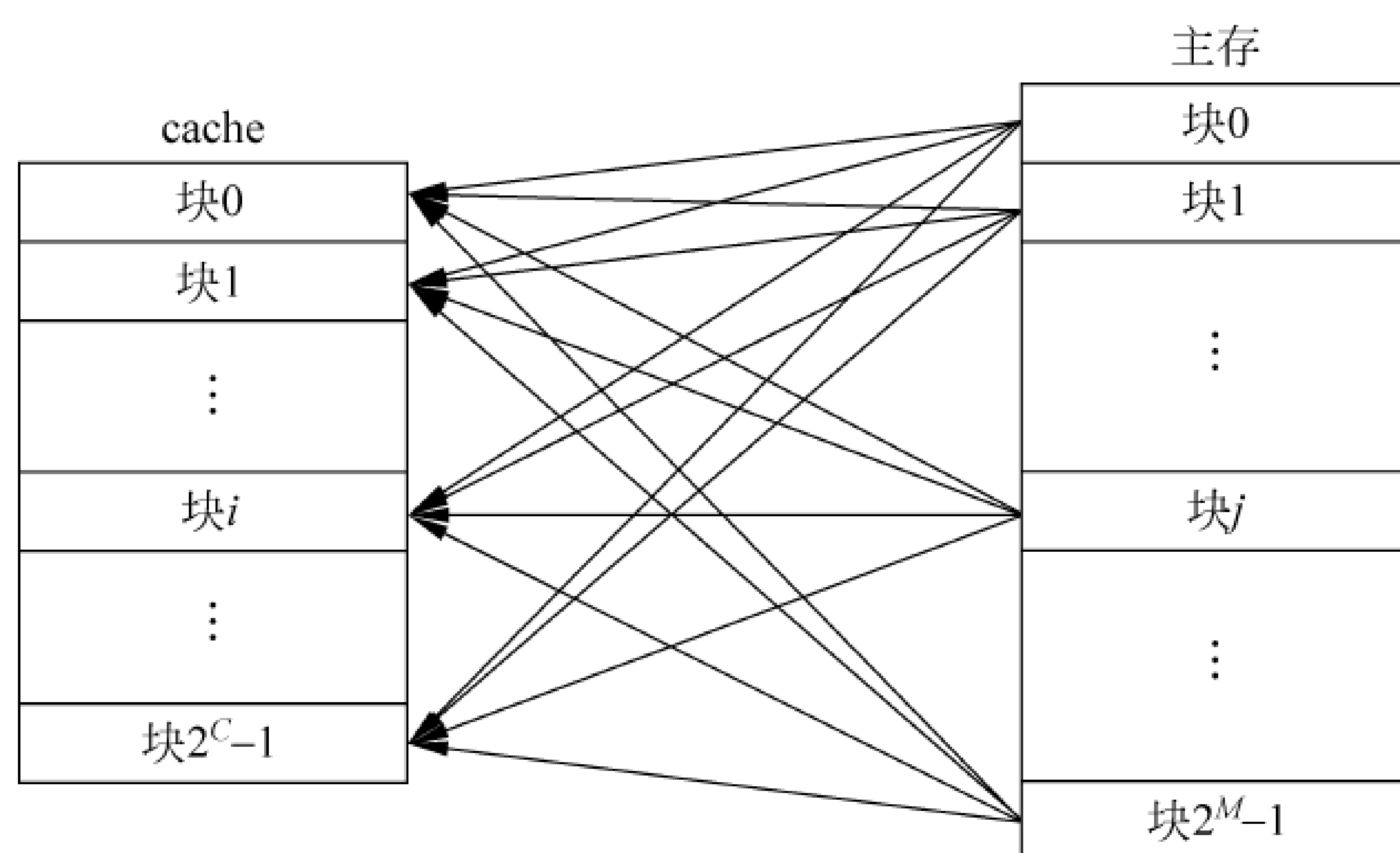


图 4-28 全相联映射方式

在全相联映射方式下，CPU 的访主存地址如图 4-29 所示。

其中，M 为主存的块号，W 为块内的字号。而 CPU 访 cache 的地址如图 4-30 所示。

其中，C 为 cache 的块号，W 为块内的字号。

图 4-29 CPU 的访存形式

图 4-30 CPU 访 cache 的地址形式

主存地址到 cache 地址的转换是通过查找一个由相联存储器实现的块表来完成的，其形成过程如图 4-31 所示。

当一个主存块调入 cache 中时，会同时在一个存储主存块号和 cache 块号映射表的相联存储器中进行登记。CPU 访存时，首先，根据主存地址中的主存块号 M 在相联存储器中查

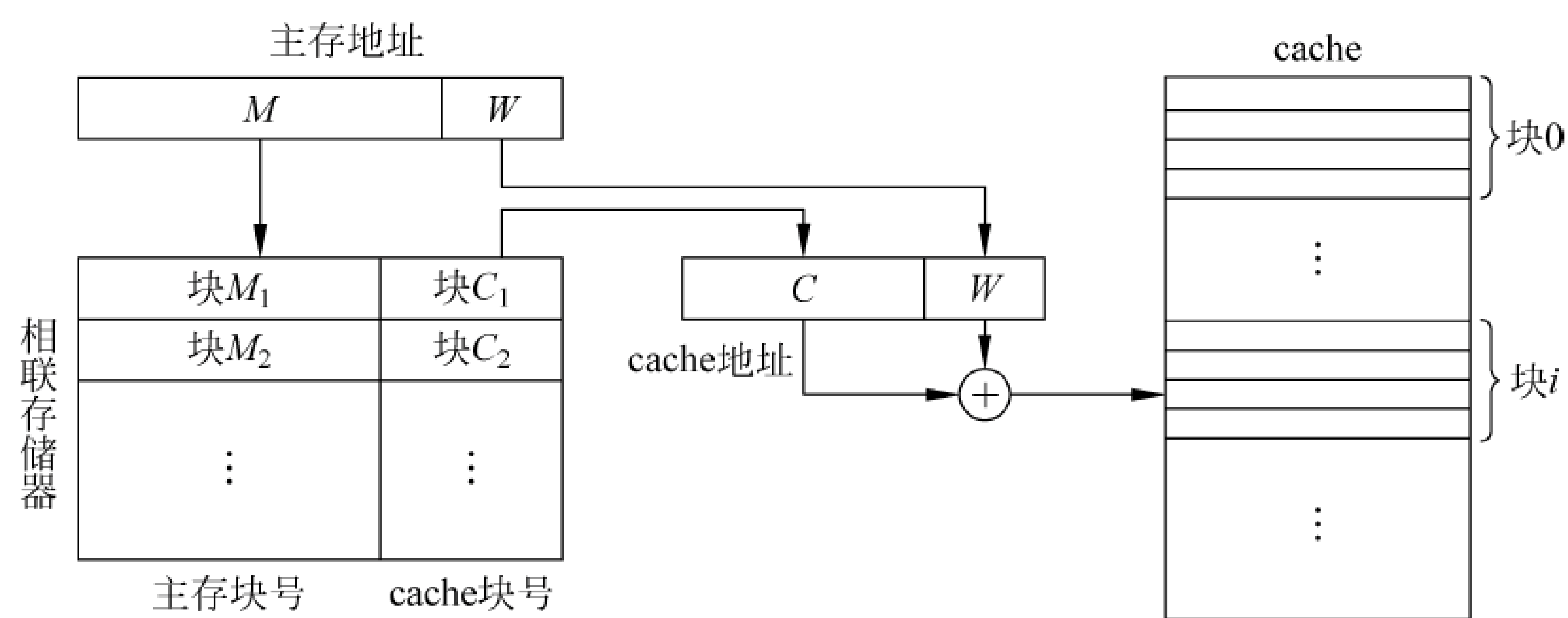


图 4-31　全相联映射的地址转换

找 cache 块号，若找到，则本次访 cache 命中，于是将对应的 cache 块号取出，并送访 cache 地址的块号 C 字段；紧接着将主存地址的块内字号 W 直接送至 cache 地址的块内字号 W 字段，从而形成一个访 cache 的地址；最后根据该地址完成对 cache 单元的访问。

全相联映射方式的优点是 cache 的空间利用率高，但缺点是相联存储器庞大，比较电路复杂，因此只适合于小容量的 cache 使用。

2. 直接相联映射

直接相联映射方式（见图 4-32）是指主存的某块 j 只能映射到满足以下特定关系的 cache 块 i 中：

$$i = j \bmod 2^C \tag{4.2}$$

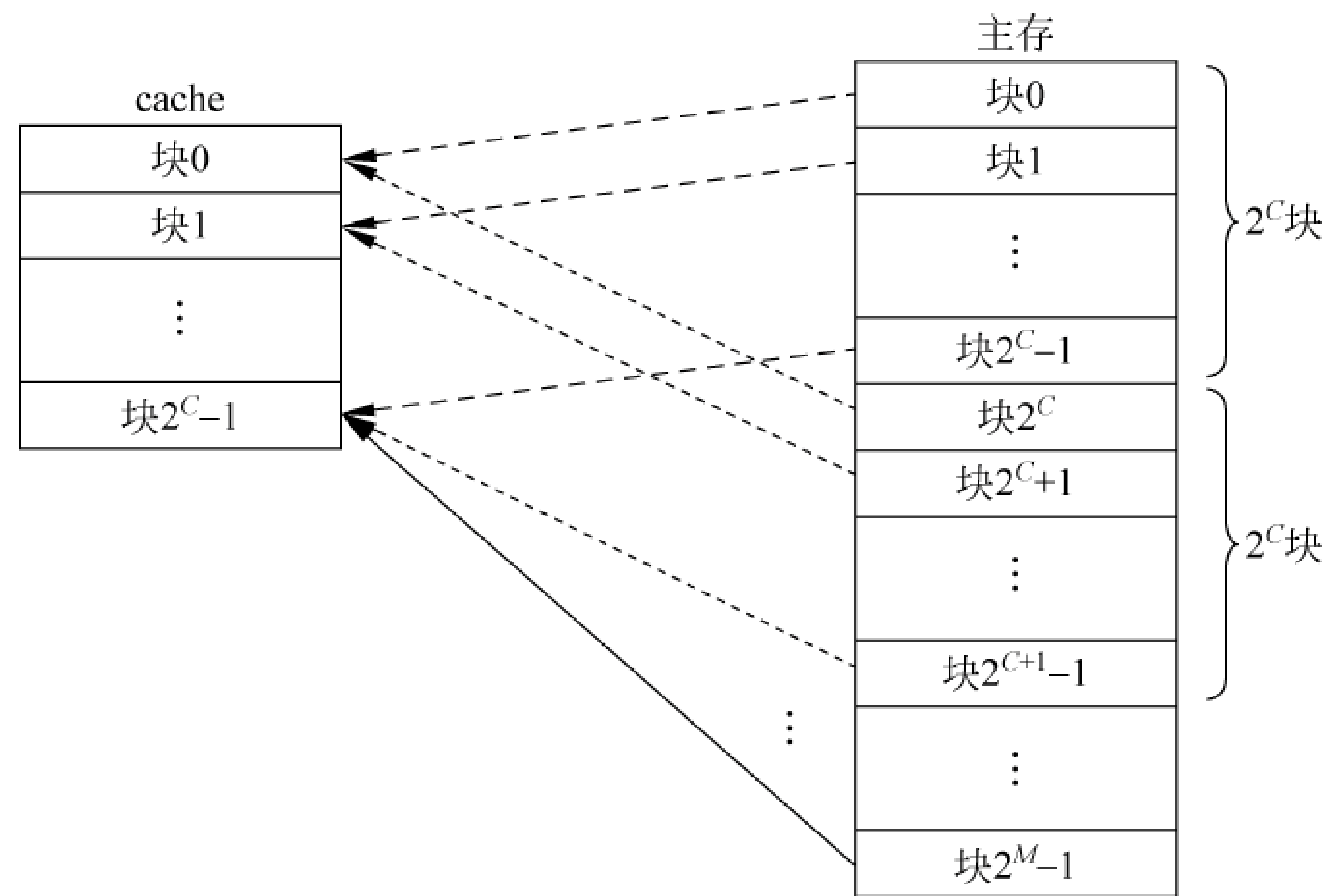


图 4-32　直接相联映射方式

图 4-32 中，主存的第 0、2^C、2^{C+1}、…、2^{M-1}块只能映射到 cache 的第 0 块，主存的第 1、第 2^C+1、第 $2^{C+1}+1$、…、2^{M-1}块只能映射到 cache 的第 1 块，…，主存的第 2^C-1、$2^{C+1}-1$、…、2^M-1 块只能映射到 cache 的第 2^C-1 块。

在直接相联映射方式下，CPU 的访主存地址如图 4-33 所示。

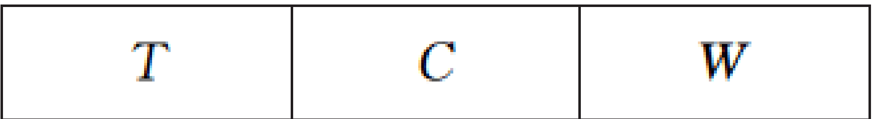


图 4-33　CPU 的访主存地址形式

其中，T 为标志号，C 为 cache 的块号，W 为块内的字号。在这里，原主存的块号 M 实际上被分成了两个字段：T 和 C，其中 C 用于指出主存的块可以映射的 cache 的块。一般来讲，主存的块数是 cache 的块数的整数倍，也就是说主存的块数 2^M 和 cache 的块数 2^C 满足关系式：

$$2^M = 2^C n$$

在直接相联映射方式下，标志号 T 是随 cache 的每个块一起存储的，其地址转换过程如图 4-34 所示。

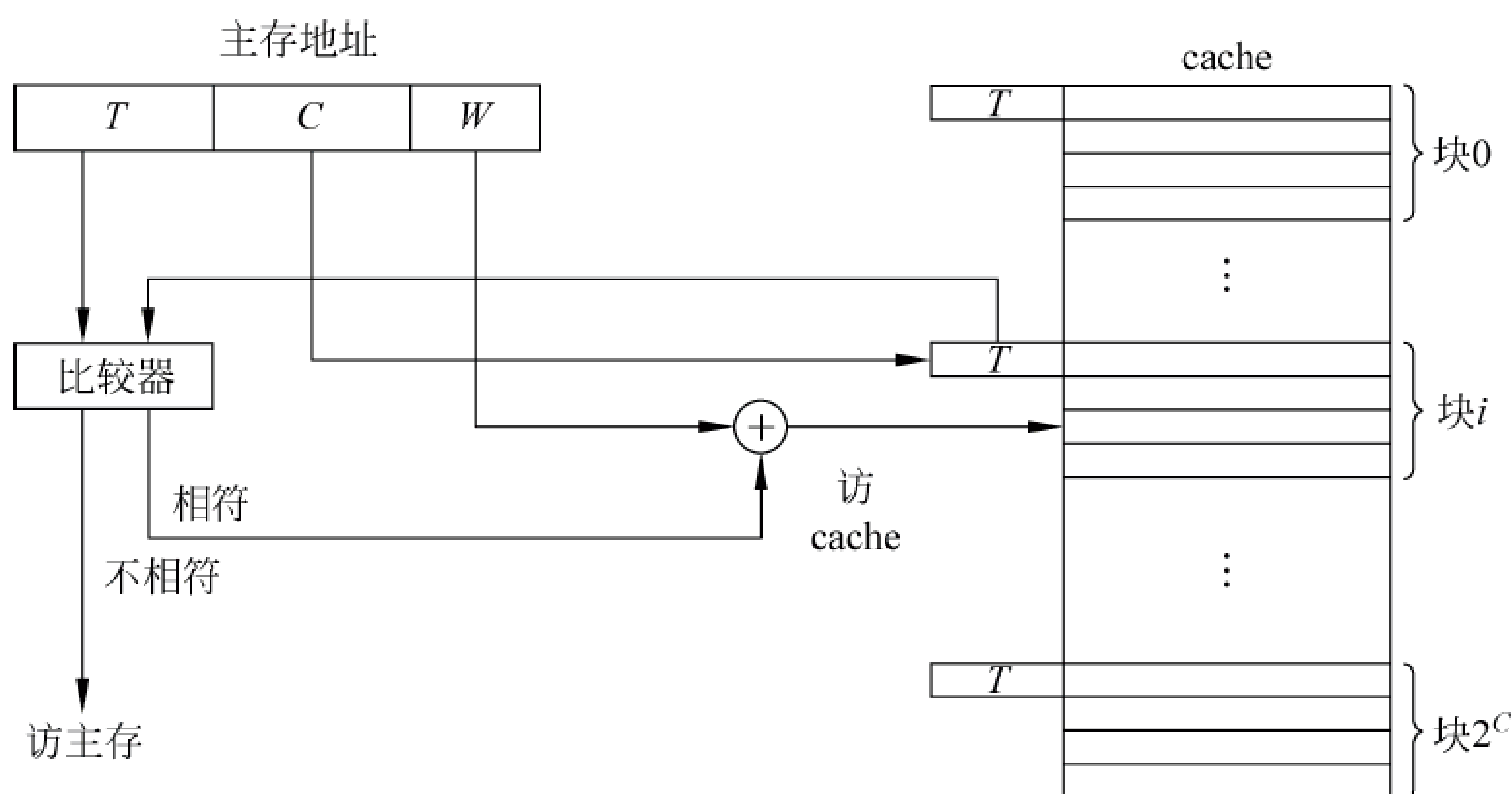


图 4-34 直接相联映射的地址转换

当一个主存块调入 cache 中时，会同时将主存地址的 T 标志存入 cache 块的标志字段中。当 CPU 送来一个访存地址时，首先，根据该主存地址的 C 字段找到 cache 的相应块，然后将该块标志字段中存放的标志与主存地址的 T 标志进行比较，若相符，说明主存的块目前已调入该 cache 块中，则命中，于是使用主存地址的 W 字段访问该 cache 块的相应字单元；若不相符，则未命中，于是使用主存地址直接访主存。

直接相联映射方式的优点是比较电路最简单，但缺点是 cache 块冲突率较高，从而降低了 cache 的利用率。由于主存的每一块只能映射到 cache 的一个特定块上，当主存的某块需调入 cache 时，如果对应的 cache 特定块已被占用，而 cache 中的其他块即使空闲，主存的块也只能通过替换的方式调入特定块的位置，不能放置到其他块的位置上。

3. 组相联映射

以上两种方式各有优缺点，而且非常有趣的是，它们的优缺点正好相反，也就是说，对于全相联映射方式来说为优点的恰是直接相联映射方式的缺点，而对于全相联映射方式来说为缺点的恰是直接相联映射方式的优点。那么，可否找到一种能较好地兼顾这两种方式的优点的映射方式呢？下面就来看看组相联映射方式。

在这种方式下，将 cache 分成 2^u 组，每组包含 2^v 块。主存的块与 cache 的组之间采用直接相联映射，而与组内的各块则采用全相联映射。也就是说，主存的某块只能映射到 cache 的特定组中的任意一块。主存的某块 j 与 cache 的组 k 之间满足以下关系：

$$K = j \bmod 2^u \tag{4.3}$$

设主存共有 $2^s \times 2^u$ 块(即 $M=s+u$),则它们的映射关系如图 4-35 所示。

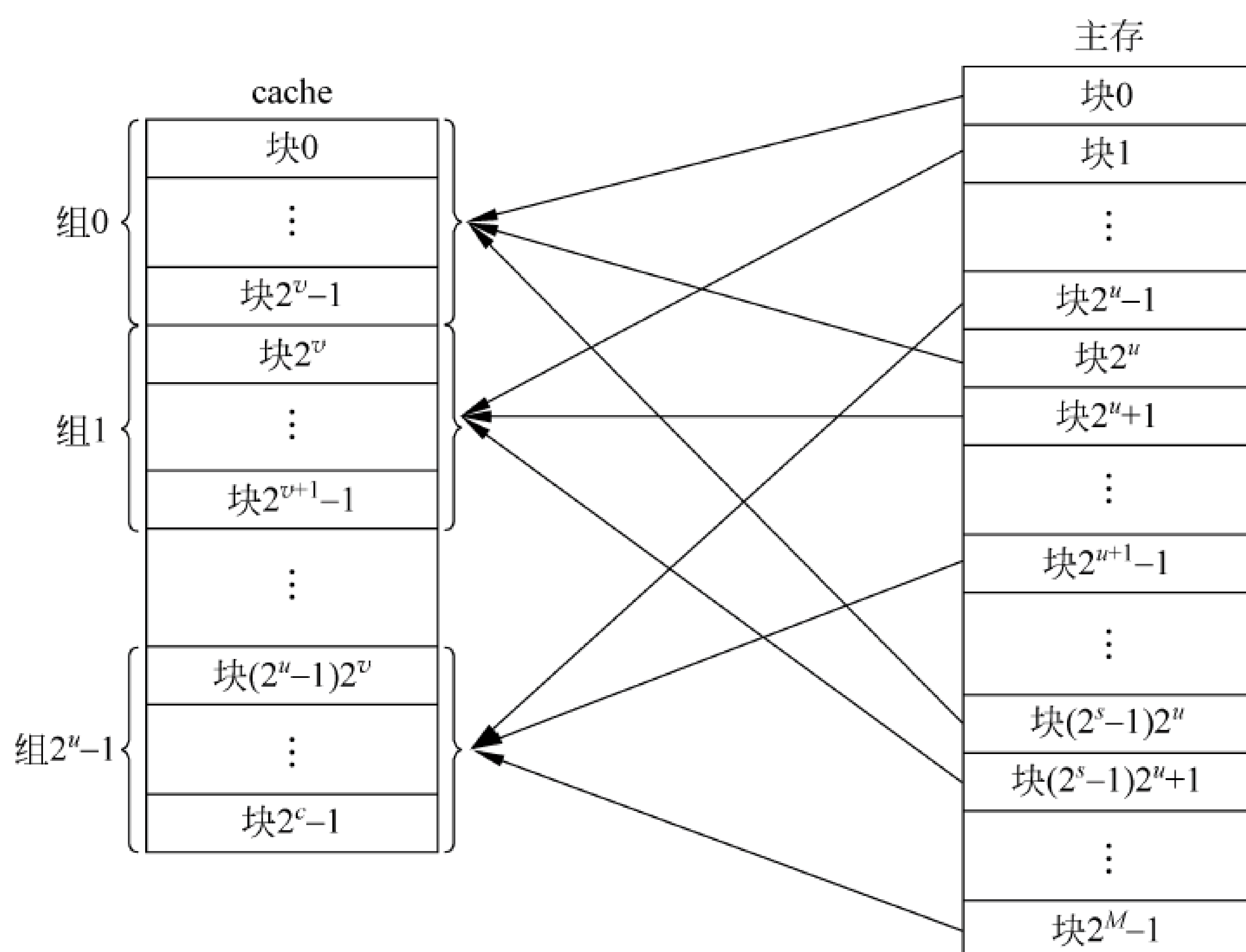


图 4-35　组相联映射方式

图 4-35 中,主存的块 0、2^u、2^{u+1}、…、$(2^s-1)2^u$ 可以映射到 cache 的第 0 组的任意一块,主存的块 1、2^u+1、$2^{u+1}+1$、…、$(2^s-1)2^u+1$ 可以映射到 cache 的第 1 组的任意一块,……,主存的块 2^u-1、$2^{u+1}-1$、…、2^M-1 可以映射到 cache 的第 2^u-1 组的任意一块。

在组相联映射方式下,CPU 的访主存地址和访 cache 地址如图 4-36 所示。

其中,u 为 cache 的组号,v 为组内的块号。cache 的块号 $C=u+v$,而主存的块号 $M=s+u$。也就是说,主存块地址的后 u 位指出了主存的这一块所能映射的 cache 的组。

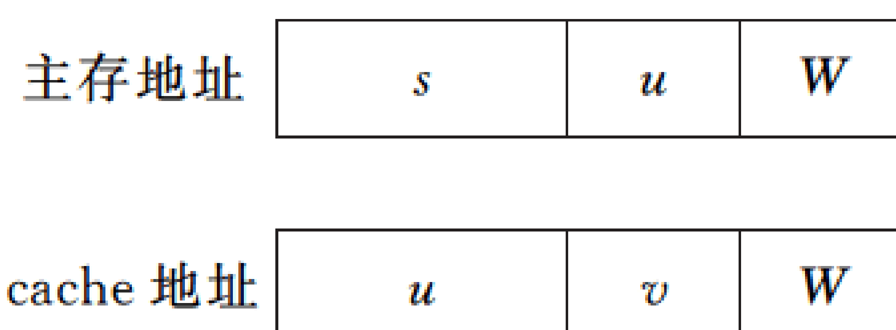


图 4-36　CPU 的访主存地址形式和访 cache 地址形式

与全相联映射方式类似的是,在组相联映射方式下,主存地址到 cache 地址的转换也是通过查找一个由相联存储器实现的块表来完成的,其形成过程如图 4-37 所示。

当一个主存块调入 cache 中时,会同时将其主存块地址的前 s 位写入一个由相联存储器实现的快表的对应 cache 块项的 s 字段中。例如,设主存的某块调入 cache 的第 1 组的第 2 块中,则在快表的组 1 第 3 项的 s 字段会登记下该主存块地址的前 s 位。

CPU 访存时,首先根据主存地址中的主存块号中的 u 字段找到快表的相应组,然后将该组的所有项的前 s 位同时与主存地址的 s 字段作比较,若相符,则说明主存块在 cache 中,于是将 cache 中该项的 v 字段取出,作为 cache 地址的 v 字段,而 cache 地址的 u、W 字段直接由主存地址的 u、W 字段形成,最后形成一个完整的访 cache 地址。当然,若比较结果没有相符项,则未命中,由主存地址直接访主存。

其实,全相联映射和直接相联映射可以看成组相联映射的两个极端情况。若 $u=0$,$v=C$,则 cache 只包含 1 组,此即全相联映射方式;若 $u=C$,$v=0$,则组内的块数等于 1,此即直

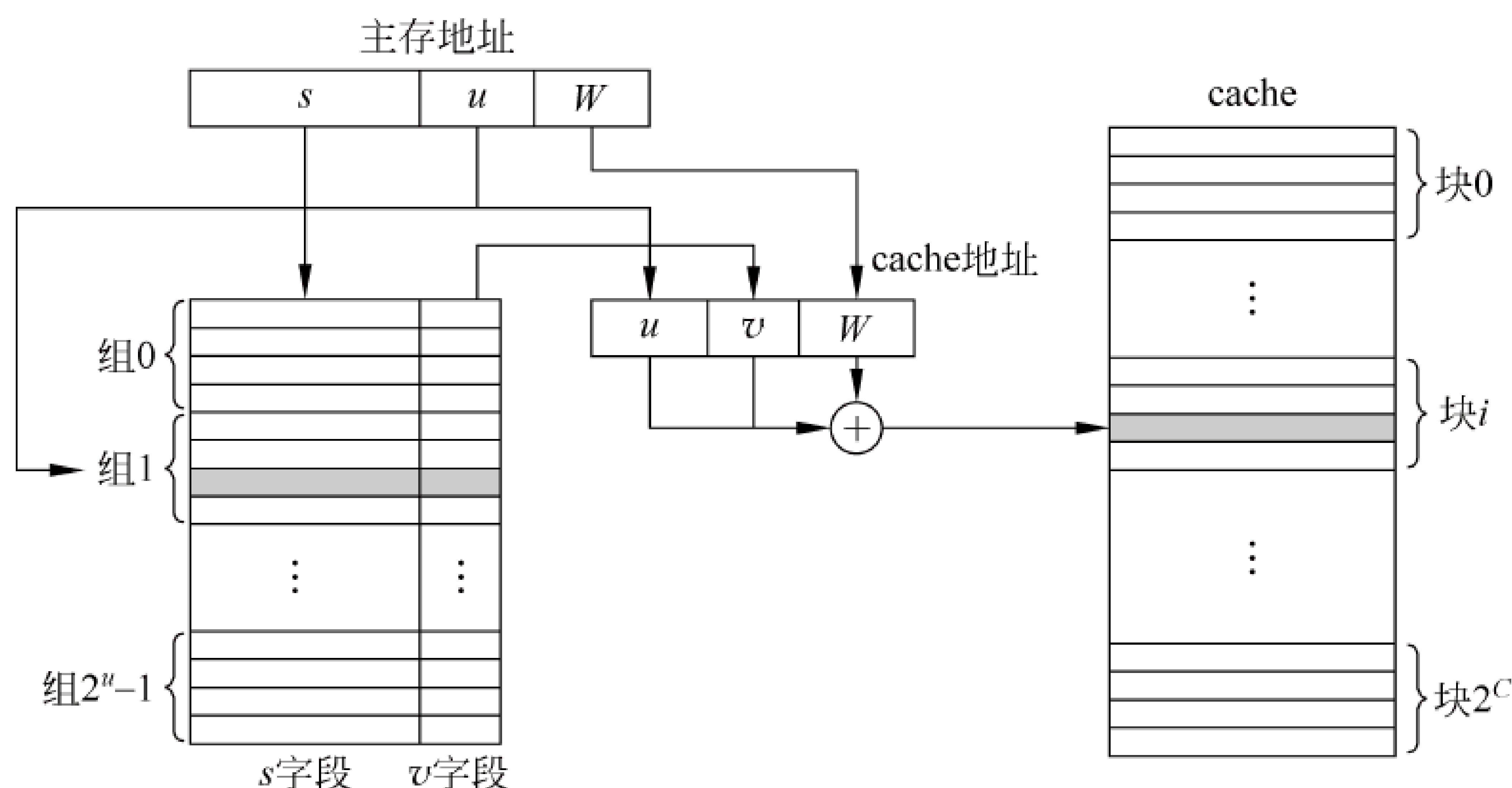


图 4-37 组相联映射的地址转换

接相联映射。

在实际应用中,相联映射方式每组的块数一般取值较小,典型值为 2、4、8、16 等,分别称为两路组相联、四路组相联等。这样一方面使得比较器的规模较小,实现较容易,例如,两路组相联采用两路比较,四路组相联采用四路比较等;另一方面,cache 每组增加的可映射块数可有效减少冲突,提高 cache 访问的命中率。

4.4.3 替换算法

当 CPU 访 cache 发生不命中的情况时,就直接访主存,并同时将所访问单元所在的一个主存块一并调入 cache 中。如果此时该主存块在 cache 中的所有特定位置已被其他块所占用,则需进行替换,即从特定位置中选择一个老的主存块从 cache 中调出,将新的主存块调入 cache。块的替换对于直接相联映射方式来说是最简单的,因为在直接相联映射方式下,主存的任意一块只能映射到 cache 的一个特定块上,因此,当需要进行替换时,只需将 cache 中的特定块替换出去即可。而对于全相联映射和组相联映射方式,情况则没这么简单,需要采取一定的替换策略或算法进行选择,主要有以下几种替换算法。

随机法:每次随机选择一个主存块替换出去。这种算法不考虑各块的使用情况,在可替换特定块数少的情况下容易造成将需使用的块替换出去,从而使 cache 的命中率会有所降低。

先进先出(FIFO)法:每次将最先调入 cache 的主存块替换出去。该算法记录每个块的调入时间,当要发生替换时,从所有可能被替换的主存块中选择一个最先调入 cache 的块替换出去。该算法实现较容易,系统开销较小,但并不十分符合主存块在 cache 中的使用规律,有可能造成正需使用的块被调出,从而影响 cache 的命中率。

最不经常使用(LFU)算法:每次将 cache 中访问最少的块替换出去。这种算法为每个块设置一个计算器,从 0 开始计数,每被命中一次计数器增 1。当需要替换时,从所有可能被替换的主存块中选择一个计数器值最小的块(最不经常使用)替换出去。这种算法将计数周期限定在对这些特定块两次替换之间的间隔时间内,因而不能严格反映近期块使用的情况。

近期最少使用(LRU)算法：每次将近期最少使用的主存块替换出去。该算法在实现时，为每个调入 cache 的主存块设置一个计数器，一个块每命中一次，其计数器清零，同时将其他块的计数器增 1。当需要替换时，从所有可能被替换的主存块中选择一个计数器值最大的块(该块近期最少使用)替换出去。这种算法符合 cache 的工作原理，可使 cache 具有较高的命中率。

对两路组相联 cache 来说，LRU 算法的实现可以简化。因为在两路组相联 cache 中，一个主存块只能调入 cache 的一个特定组的两块中的一块，因此只需设置一个标志位，当两块中的一块(假设为 A 块)被命中时，标志位置 1，而另一块(假设为 B 块)被命中时，标志位清 0。当需要替换时，置需检查此标志位的状态即可：为 0 替换 A 块，为 1 替换 B 块。Pentium CPU 内的数据 cache 采用的是一个两路组相联结构，使用的就是这种简化的 LRU 替换算法。

下面以图 4-38 的例子，简单说明 FIFO 和 LRU 替换算法的工作原理和命中率情况。

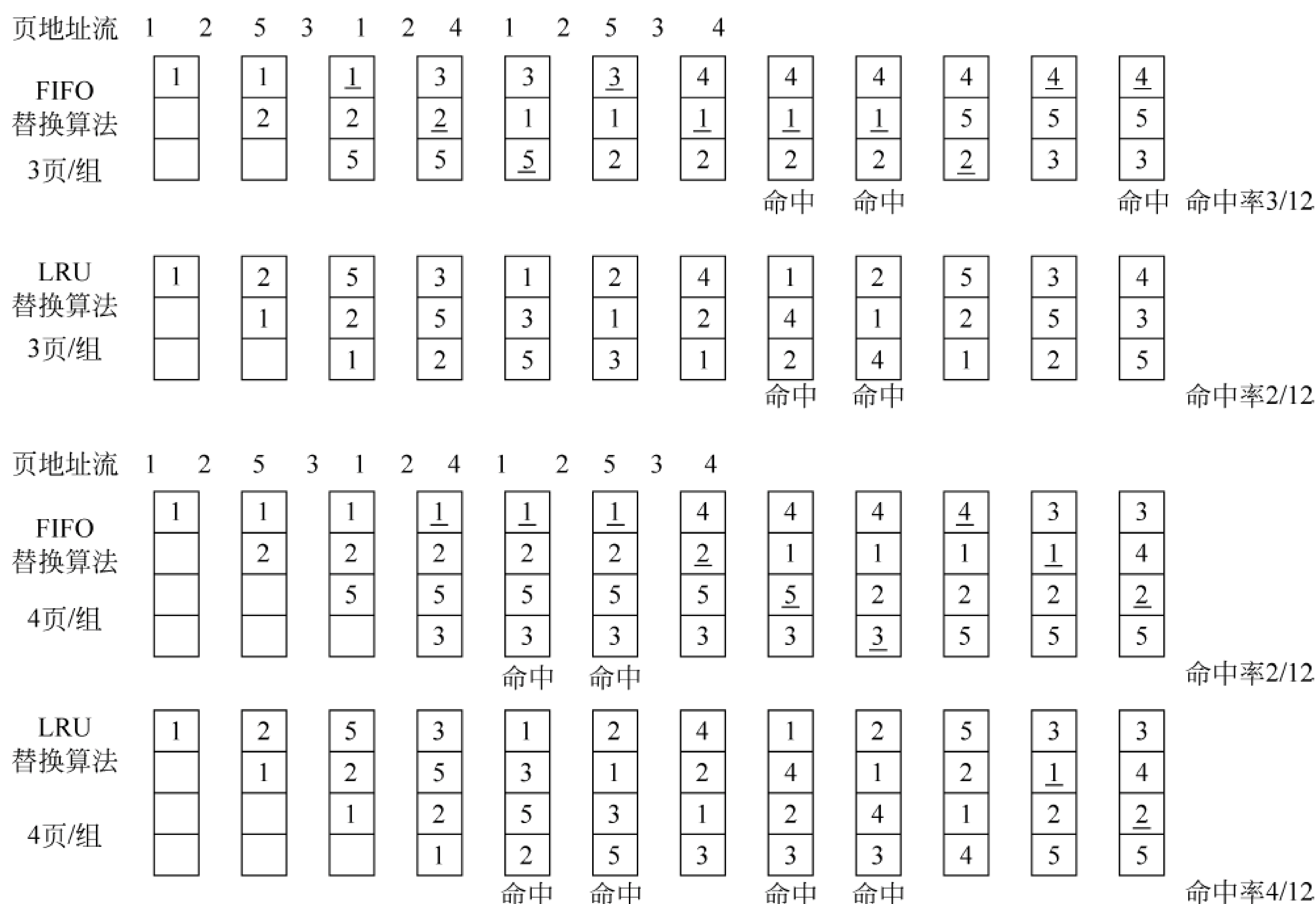


图 4-38　FIFO 和 LRU 替换算法的命中率情况

从上面的例子可以看出，LRU 算法的平均命中率要高于 FIFO 算法，而且随着分组容量的增大，LRU 算法的命中率必定提高，FIFO 算法则未必。因为在 LRU 算法中，同一时刻小容量分组之页的集合必定是大容量分组之页的集合的子集，因而小容量分组的命中点必定还是大容量分组的命中点，这种特征算法称为堆栈算法，而在 FIFO 算法中，同一时刻小容量分组之页的集合则不一定是大容量分组之页的集合的子集。

4.4.4　cache 的写策略

cache 中的内容可以看成主存的一个子集，即每次总是将主存的部分块的内容调入

cache 中。CPU 对 cache 的读操作是不会影响 cache 的内容的，也就不会影响 cache 与主存内容的一致性。但写就不一样了，当 CPU 对 cache 的某块进行了一次写操作，但对主存该块的内容却没有同时进行写时，会造成主存和 cache 内容的不一致。因此，对 cache 的写需要采取某种策略或方法。

全写法（Write-Through）：又称写直达法，CPU 每次在写 cache 命中时，在写 cache 的同时，也对相应的主存块进行写入；当写 cache 未命中时，则直接写主存。全写法是写 cache 和写主存同步进行的，其优点是 cache 和主存的内容能保持高度的一致性，缺点是 cache 对 CPU 的写操作起不到高速缓存的作用，失去了 cache 的功效。统计表明，一般程序中的写操作占到了存储器操作的 15%左右，对于一些特殊应用的程序，其写操作比例会更高。在这种情况下，采用全写法实现写操作会影响 cache 的效率。

写回法（Write-Back）：CPU 每次在写 cache 命中时，只写 cache，暂不写主存，只有当某被写命中的块从 cache 中替换出去时才写回主存。这种方法使 cache 在 CPU 的写操作中也同样能发挥高速缓存的作用，但却存在主存与 cache 内容不一致的隐患。写回法在实现时，为每个 cache 块设置一个标志位，以反映 cache 的某块是否被修改过。当某块被替换出去时，根据标志位决定在替换的同时是否进行回写操作。未被修改过的块在替换出去时，无须进行回写，只有被修改过的块在替换出去时，才需进行主存的回写。

现在越来越多的机器都采用了二级 cache，在处理写操作时需要考虑的不仅仅是写回主存，还需考虑写回二级 cache。写一次法（Write One Time）是采用了二级 cache 的机器中常用的写操作策略，其具体方法是：写命中时的处理方法和写回法基本相同，只是第一次写命中时要同时写入主存和其他二级 cache 等。奔腾 CPU 的片内数据 cache 就采用了这一方法。

4.4.5 cache 性能分析

1. CPU 访存时间的分析

在 CPU 与主存之间增加 cache 的目的，就是要使 CPU 的访存速度接近于访 cache 的速度。那么，在怎样的情况下能做到这一点呢？下面进行一下分析。

设 N_c 表示某一段时间 cache 完成存取的总次数，N_m 表示这一段时间主存完成存取的总次数，h 定义为命中率，则有

$$h = \frac{N_c}{N_c + N_m} \tag{4.4}$$

再假设 T_c 为 cache 存储器的读写周期时间，T_m 为主存储器的读写周期时间，则 cache-主存系统的平均访问时间 T_A 为

$$T_A = hT_c + (1-h)T_m \tag{4.5}$$

从式(4.5)可以看出，平均访问时间与命中率是紧密相关的，命中率越高，平均访问时间越接近 cache 的读写周期时间。

系统的目标是以较小的硬件代价使得 cache-主存系统的平均访问时间 T_A 越接近 T_c。设 $r = T_m/T_c$ 表示主存慢于 cache 的倍率，则 cache-主存系统访问效率 e 为

$$e = \frac{T_c}{T_A} = \frac{T_c}{hT_c + (1-h)T_m} = \frac{1}{r + (1-r)h} \tag{4.6}$$

【例 4.3】 设 CPU 执行某一段程序时共进行了 10 000 次的访存操作，其中命中 cache 并完成访 cache 的操作共 9800 次，而未命中访主存的次数为 200 次，已知 cache 的读写周期为 20ns，主存的读写周期为 80ns，求执行这一段程序时该 cache-主存系统的平均访问时间。

解： 命中率 $h=9800/10\,000=0.98$，则 cache-主存系统的平均访问时间为

$$\begin{aligned} T_A &= hT_c + (1-h)T_m \\ &= 0.98\times 20 + (1-0.98)\times 80 \\ &= 21.2(\text{ns}) \end{aligned}$$

表 4-1 列出的是 $T_c=10\text{ns}$，$T_m=60\text{ns}$ 时，不同 h 值所对应的平均访问时间 T_A 不同。

表 4-1　命中率和平均访问时间

h	T_A
0.0	60ns
0.1	55ns
0.2	50ns
0.3	45ns
0.4	40ns
0.5	35ns
0.6	30ns
0.7	25ns
0.8	20ns
0.9	15ns
1.0	10ns

2. 块的大小

主存和 cache 都划分了同样大小的块，当 CPU 访问 cache 失效时，需将一个主存块调入 cache。那么，一个块的容量该多大算合适呢？显然，当块容量从很小逐渐变大时，会使 cache 的命中率增加。这是因为，当 CPU 所访问的一个字单元不在 cache 中时，会从主存中将该单元以及与前后若干单元组成的块一并调入主存，块越大，CPU 访问完该单元后再访问其他单元的命中率自然就会高。但随着块容量的继续增大，对命中率增加的贡献将会越来越小。因为随着块的增大，离本次访问未命中的字单元越来越远的一些字单元也被调入 cache，而这些单元与刚访问过的单元并不在一个程序空间的“局部”，也就是说不会在同一个块中访问到它们，它们本次调入的意义不大。

块的容量大小与命中率的关系是比较复杂的，它取决于不同程序的局部性特征，对所有程序而言，很难确定一个最优的块的大小。通常认为块容量大小在 8～64B 是比较合适的。

3. cache 的数量

最初构建主存-cache 层次时，只设置了一个 cache。而近些年来，越来越多的机器采用了多级 cache 和指令缓存与数据缓存分离等技术，进一步提高了 CPU 的访存效率。

随着集成度的提高，现代的 CPU 在芯片中集成了一个小容量的 cache，与 CPU 外部的 cache 一起构成二级 cache 系统，其中，CPU 内部的 cache 为第 1 级(L1)，CPU 外部的 cache 为第 2 级(L2)。一般来讲，L1 cache 比 L2 cache 容量更小，速度更快。“L1 cache-L2 cache-主存”这种层次从工作原理上讲与前述的 cache 工作原理是完全相同的，即 CPU 首先访 L1 级 cache，若不命中，再访问 L2 级 cache 和主存。

与通过外部总线连接的外部 cache 相比，CPU 的片内 cache 与 CPU 的路径更短，可减少处理器在外部总线上的操作时间，因而加快了 CPU 的访存速度，进一步提高了系统的性能。

近些年来，对 cache 系统设计的另一种趋势是采用分立 cache 技术，也就是将指令和数据分开，分别存放在指令 cache 和数据 cache 中。这种分立 cache 技术有利于 CPU 采用流水线方式执行指令。在流水线中，往往会发生在同一个操作周期同时需要预取一条指令和

执行另一条指令的取数据操作的情况。若采用指令和数据统一的 cache,则这种情况会造成取指令和取数据的访存冲突,冲突的结果就是使得流水线产生断流的情况发生,从而严重影响流水线的效率。采用分立 cache 技术,因为取指令和取数据分别在不同的 cache 中同时进行,因而不会产生冲突,有利于流水线的实现。

4.4.6 cache 举例:Pentium 4 的 cache 组织

首先通过表 4-2 来看看 Intel CPU 使用 cache 技术的演变。80386 不包含片内 cache,但首次在外部(机器主板上)使用了 cache 技术。80486 处理器速度的提高使外部总线成为处理器访问外部 cache 的瓶颈,于是在 486 片内使用了一个小容量的 cache(8KB),采用每块 16B 的四路组相联映射机制。Pentium 处理器采用了分立 cache 技术,包含两个片内 L1 cache,分别用于数据和指令。Pentium Pro 处理器采用了前端总线技术,处理器和 L2 cache 之间采用高速总线互联,而 Pentium Ⅱ 处理器则将 L2 cache 移到了片内,其容量达到了 256KB,每块 128B,采用 8 路组相联映射机制。Pentium Ⅲ 处理器添加了一个外部 L3 cache,而 Pentium 4 处理器则将大容量 L3 cache 移到了片内。

表 4-2 Intel CPU 使用 cache 技术的演变

Intel CPU	采用的 cache 技术	解决的问题
80386	使用外部 cache	外部存储器慢于系统总线
80486	将外部 cache 移到处理器芯片上	处理器速度的提高导致外部总线成为 cache 存取的瓶颈
	采用外部 L2 级 cache	由于片上可用面积有限,内部 cache 相当小
Pentium	使用分立的数据 cache 和指令 cache	当指令预取部件和执行单元同时请求访问 cache 时产生冲突,使指令执行的流水操作产生断流
Pentium Pro	采用分立的前端总线(外部的主总线)和后端总线 BSB 技术,BSB 以更高速度运行,专用于 L2 cache	处理器速度的提高导致外部总线成为 L2 cache 存取的瓶颈
Pentium Ⅱ	将 L2 cache 移到处理器芯片上	
Pentium Ⅲ	添加外部的 L3 cache	某些应用与海量数据库打交道,必须具有对海量数据的快速存取能力,而片上的 cache 容量太小
Pentium 4	将大容量的 L3 cache 也移到处理器芯片上	

图 4-39 给出了一个 Pentium 4 处理器的内部组织结构简化图,着重 cache 的组织和布局。Pentium 4 处理器的核心由以下几个主要部件组成。

(1) 取指令、译码部件(Fetch/Decode Unit):根据指令地址从 L2 cache 中取指令,并将指令代码译码成一系列的微操作存入 L1 指令 cache 中。

(2) 执行单元(Execution Unit):执行 L1 指令 cache 中的微操作,并由 L1 数据 cache 取所需数据,存入寄存器中暂存。执行单元可依据数据相关性和资源可用性调度微操作的执行,使得微操作的执行可按不同于所取指令流的顺序被调度执行,只要时间许可,此单元完成将来可能需要的微操作的推测执行。

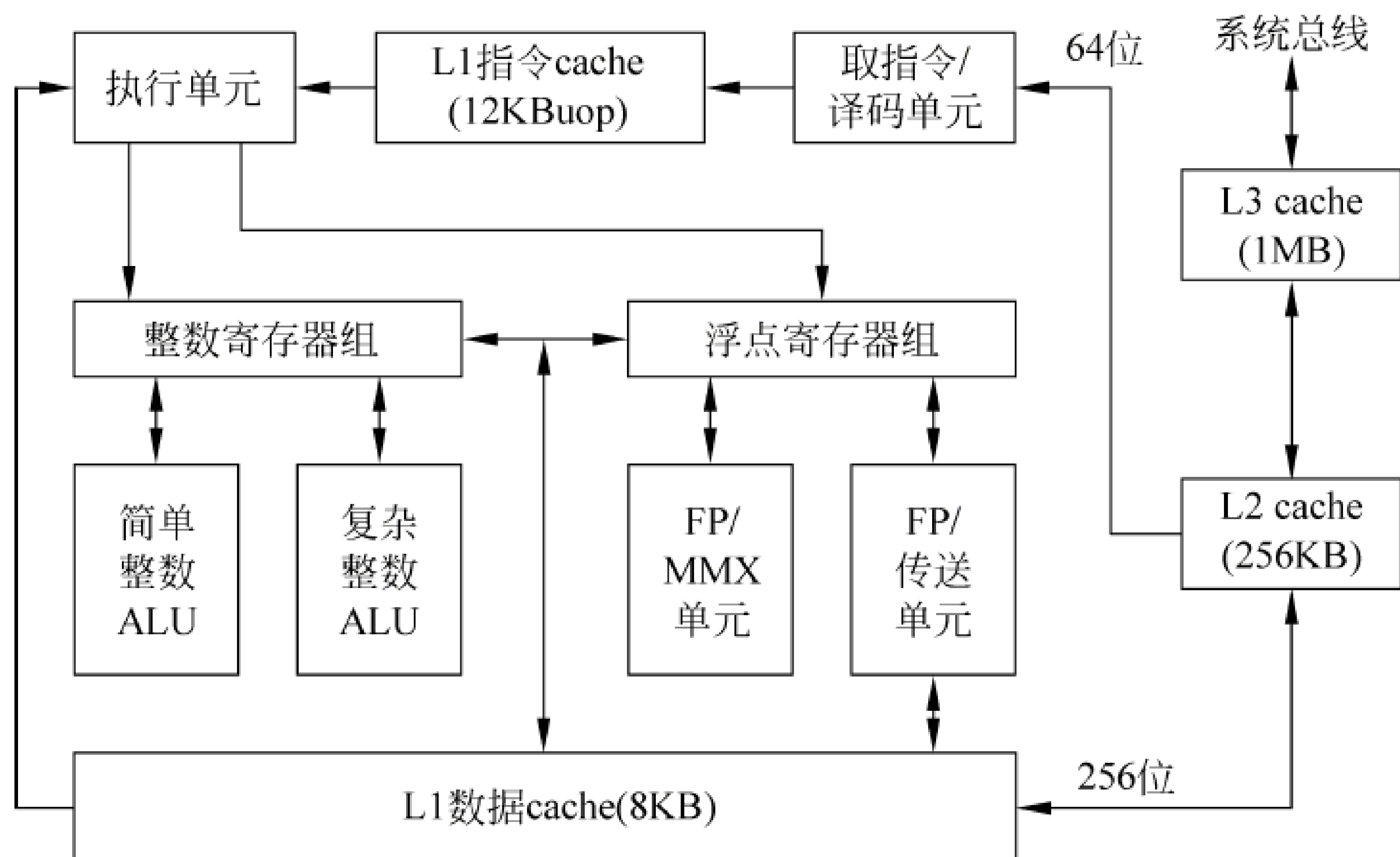


图 4-39 Pentium 4 处理器的内部组织结构图

(3) 存储器子系统(Memory Subsystem):这部分包括 L2 和 L3 cache 及系统总线。当 L2 和 L3 cache 未命中时,使用系统总线访问主存。系统总线还用于访问系统输入输出资源。

(4) 整数运算和浮点数运算部件 ALU。

不同于所有以前的 Pentium 和其他大多数处理器所采用的 cache 组织,Pentium 4 指令 cache 位于指令译码器和执行单元之间。取指令/指令译码部件完成指令的预取和译码工作,L1 cache 存储实际上是指令译码后得到的微操作码,这样就使得执行部件能从 L1 cache 中取出直接执行的微代码,加快了指令的执行速度。

L2 和 L3 cache 采用的是 8 路组相联映射机制,每块 128B。

4.5 虚拟存储器

早在 1961 年英国曼彻斯特大学的 Kilburn 等就提出了虚拟存储器的概念,经过 20 世纪 60 年代初到 70 年代初的发展完善,虚拟存储器已广泛应用于大中型计算机系统中。1982 年,Intel 公司首次在其推出的微处理器 80286 上集成了虚拟存储器管理功能。到现在,几乎所有的机器都采用了虚拟存储器技术。

4.5.1 虚拟存储器实现的基本原理

在前面的 4.1.3 节中已经讲到,现代计算机的存储器采用分层结构,即“cache-主存-辅存”这种三级结构。实际上,这里包含了两个主要层次:一是“cache-主存”层次,其目标是通过增加小容量的高速 cache 提高 CPU 的访存速度,但对程序空间的容量不产生任何影响;二是“主存-辅存”层次,其目标是为用户提供一个远远大于主存容量的程序空间。

随着用户程序对主存容量的要求越来越高,计算机主存储器在容量配置上不断提高,从早期的几十 KB 到几百 KB 到现在的几百 MB 到几十 GB。但主存容量的配置总是受到一定因素的制约。一方面,出于机器性能价格比的考虑,价格较贵的主存储器容量不可能配置

得太大。其实，作为计算机系统设计者来讲，如果只追求性能，不考虑价格，机器的主存储器完全可以使用单一的存储介质实现，即采用大容量、高速的存储器件构成计算机的主存储器。但这样做的后果是机器的价格很高，其中存储器占到了机器成本的主要部分；另一方面，在不同的应用场合，应用程序对主存容量的要求有所不同。作为通用计算机来讲，满足了大容量存储器要求的应用需要，对小容量存储器要求的应用就显得浪费。在既要考虑性能又要考虑价格的情况下，如何权衡和配置计算机的主存储器就显得非常困难。如果能够通过合理的设计，在不增加机器太多成本和开销的情况下，为用户程序提供一个足够大的存储空间，从而满足所有应用的需要，何乐而不为呢。虚拟存储器正是为实现这一目标而提出的。

虚拟存储器是一种由价格较高、速度较快、容量较小的主存储器和一个价格低廉、速度较慢、容量巨大的辅助存储器组成的存储层次，在系统软件和辅助硬件的管理下就像一个单一的、可直接访问的大容量存储器，以透明方式为用户程序提供一个远大于主存容量的存储空间。

在未使用虚拟存储器的机器中，用户程序所需存储空间不能大于机器实际能提供的主存容量，如果用户程序过大，其所需存储空间超出了主存能提供的容量，则要么该程序无法运行，要么由用户自行完成对程序的分块处理和存储分配工作，这对一般用户来讲是非常困难的。而在使用了虚拟存储器的机器中，允许用户程序空间大于机器实际主存空间。当用户程序所需存储空间大于主存当前能够提供的存储容量时，由系统将用户程序的一部分先存放在辅助存储器中，当运行到这部分程序时，再将它们从辅存中调出，并分配给它们合适的主存空间执行。程序在存储器中的分配完全在系统的控制下进行，无须用户的干预。这对多用户、多道程序系统来讲显得尤为重要，可以避免因为用户的自行存储分配影响到其他用户程序在存储器中的正常执行的情况发生。

虚拟存储器的“主存-辅存”层次的实现原理与前面讲述的“cache-主存”层次的实现原理很相似。这主要体现在两个方面：一是组织结构相似，在这两种层次中，面向 CPU 的存储器是由相对来讲小容量高速存储器件构成的，而后援存储器则是由相对来讲大容量低速存储器件构成的，小容量存储器中存储的内容总是大容量存储器所存储内容的一部分；二是工作原理相似，用户程序事先存放在大容量后援存储器中，当执行程序时，CPU 首先访问的是小容量存储器，若命中，则完成本次访问，若不命中，则再到大容量存储器中访问，并同时按一定的规则将大容量存储器中的一部分内容替换小容量存储器中的一部分内容。当然，还有非常重要的一点就是实现虚拟存储器同样基于的是程序的局部性原理，若没有程序访问的局部性，CPU 访问小容量存储器频频失效，则会大大降低 CPU 访问存储系统的效率。

在实现虚拟存储器管理的机器中，能提供给用户程序运行的是一个比主存大得多的虚拟空间，称为虚存；而真正为用户程序提供实际运行空间的是主存，又称为实存。程序中出现的指令或数据地址是逻辑地址，又称为虚地址；而主存地址是真正的物理地址，又称为实地址。与“cache-主存”层次相似，当 CPU 访问虚拟存储器时，首先给出的是虚地址，通过一定的地址转换机制，变换成一个实地址，再按此实地址访问主存。虚拟存储器的工作原理如图 4-40 所示。

从图 4-40 中可以看出，与“cache-主存”层次相似，在实现虚拟存储器管理中，要完成一个虚地址到实地址的变换过程，这一变换过程是由一个存储管理单元(MMU)来实现的。另外，当访主存失效(不命中)时，需要将虚存中的一部分内容调入主存中，调入的容量大小

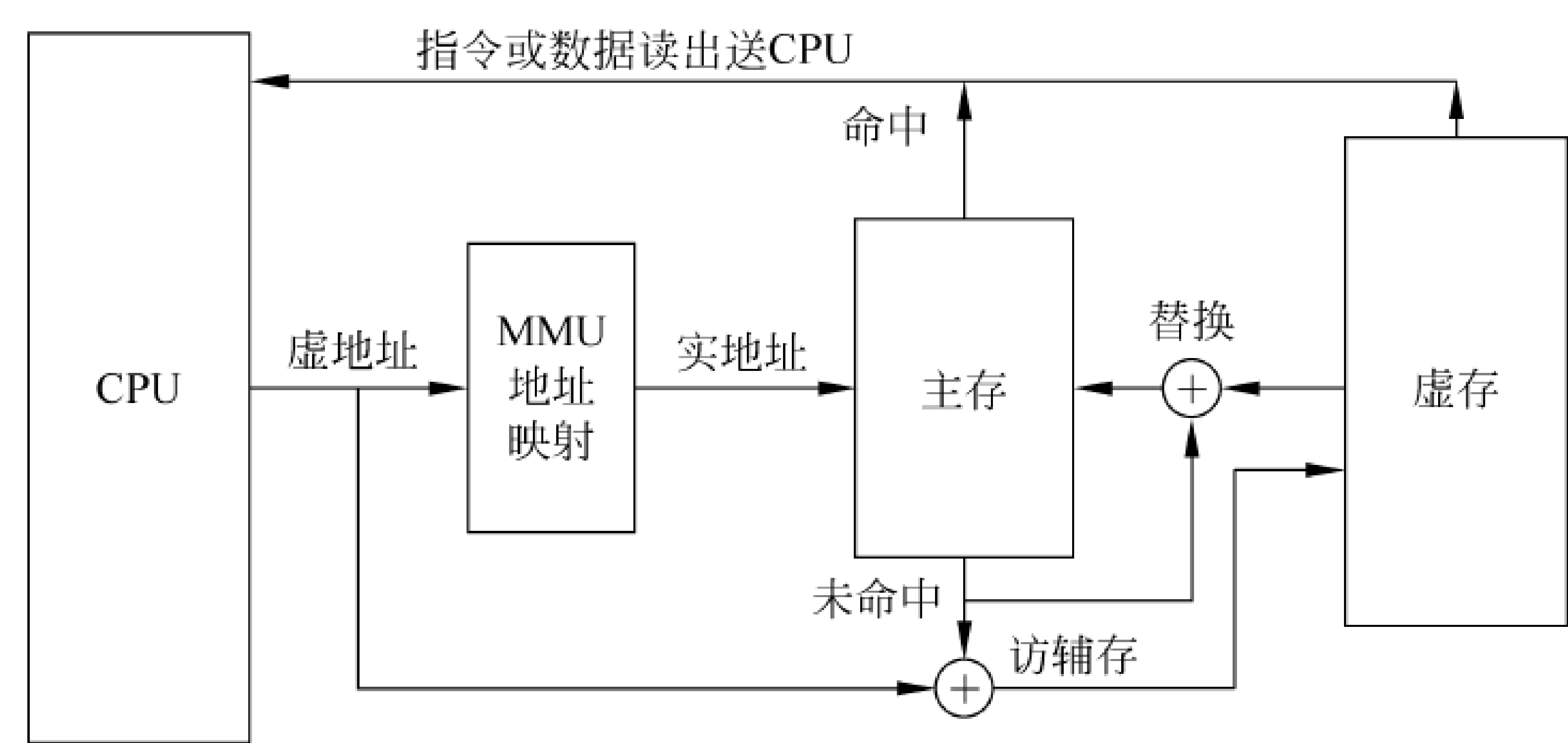


图 4-40 虚拟存储器工作原理图

也同样与虚拟存储器管理机制有关。在这里主要介绍虚拟存储器的三种管理机制：分页式虚拟存储器管理、分段式虚拟存储器管理和段页式虚拟存储器管理。

4.5.2 虚拟存储器的分页式管理

在采用分页式管理的机器中，主存和虚存都划分成固定大小的块——页，主存的页称为实页，虚存的页称为虚页，实页和虚页的大小是相同的。实地址和虚地址都由两部分组成，分别为实页号/虚页号和页内偏移地址，如图 4-41 所示。

对于 CPU 访问的某一个字单元来说，实地址中的实页号和虚地址中的虚页号是不同的，但两个地址中的页偏移是相同的。

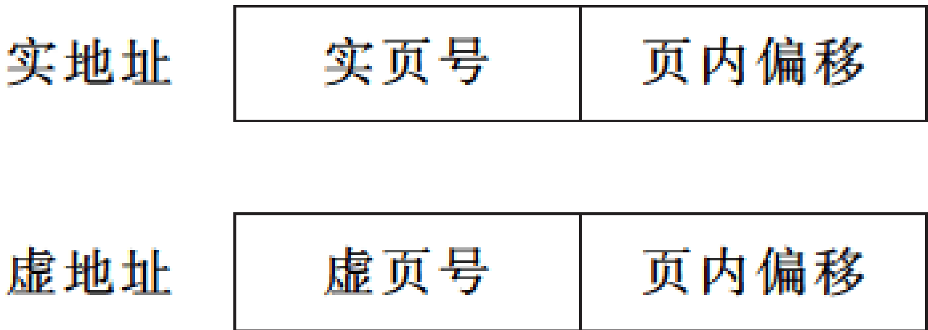


图 4-41 实地址和虚地址的组成形式

在分页式虚拟存储器中，任一个虚页可以映射到主存的任一个实页上，虚地址到实地址的变换是通过存放在主存中的页表来实现的。页表由一个个页表项组成，每个虚页在页表中占一个页表项，并按虚页号的顺序排列。页表项的内容主要包括该虚页所在主存的实页号和一些控制信息，如图 4-42 所示。

F：装入位。*F*=1 表示该虚页已装入主存中；*F*=0 表示该虚页还没装入主存。

C：修改位。用于表示该页装入主存后是否被修改过。

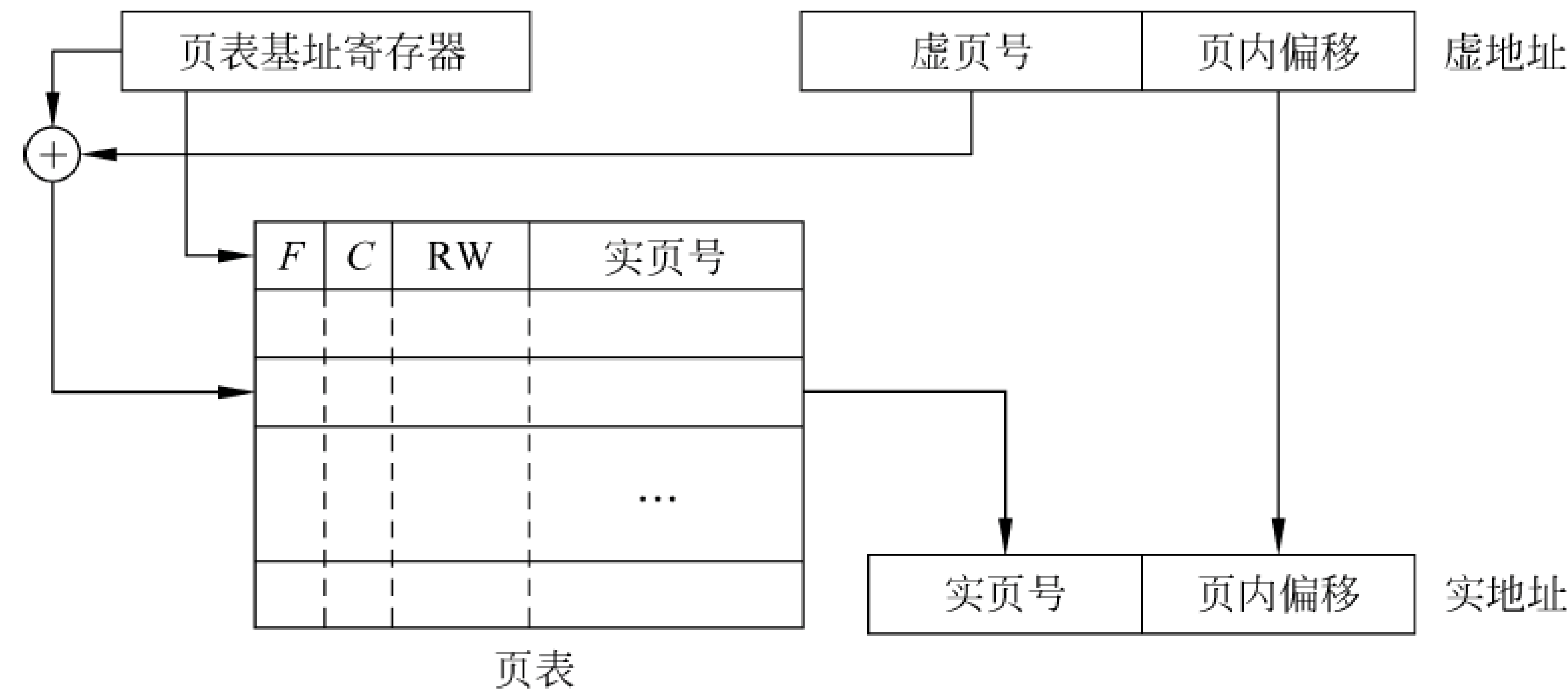


图 4-42 分页式虚拟存储器地址变换

RW：访问方式字段。用于表示该页可以允许以“读-写-执行”方式中的哪些方式访问。

页表在主存中的位置通过一个页表基址寄存器来指明，页表项的内容是在每个虚页装入主存时填写的。

当用户程序中给出一个访存虚地址时，系统首先以页表基址寄存器中的内容为基地址，虚地址中的虚页号为偏移量，找到该虚地址所对应的存放在主存中的页表的页表项。然后根据装入位 F 的状态判断该虚页是否已经装入主存。若已装入，则从该页表项中取出实页号，再结合页内偏移地址一起共同形成一个访主存的物理地址。若未装入，则产生页面失效（故障），启动输入输出子系统，将该虚地址所在的页从辅存中调入主存。

由于页面的划分只是机械地对虚、实存空间进行等分，因此，页面失效有可能发生在一条指令的执行过程之中。例如，对于按字节编址的存储器，就可能出现一条指令跨页存储的情况。当前一页已在主存，而后一页不在主存时，则在取指令过程中就会发生页面失效。同理，在取操作数（特别是字符串）、间接寻址及写回结果的过程中都可能发生页面失效。对于这种故障，处理机必须立即响应和处理，否则该条指令无法执行下去。如何保存和恢复故障点的现场，使得故障处理完成后，又能正确地从断点处继续执行指令，这是保证虚拟存储器能否正确工作的关键性问题之一。目前，有的机器采用后援寄存器技术，即把该条指令的故障现场全部保存下来，当处理完该故障，把所需要的页调入主存后，再从故障点处继续执行完该条指令。有的机器则是保存部分有关现场，使该指令能从头再开始执行。有的机器则采用预判技术。例如，在执行字符串指令前，预判断字符串的首、尾字符所在页是否已在主存中，如果在，则执行这条指令，只要有一个还没装入主存，就产生页面失效故障请求，在把该页调入后，才开始执行这条字符串指令。

在页面失效时，要求把该页由辅存调入主存，因此必须给出该页在辅存的物理地址。辅存一般按信息块（对磁盘而言为扇区）编址，且一个块的大小通常等于一个页面的大小。对磁盘存储器来说，其物理地址格式如图 4-43 所示。

盘机号	柱面号	磁头号	扇区号

图 4-43　磁盘存储器的物理地址格式

将虚地址变换成磁盘的物理地址时，即把虚页号变换成某个磁盘机上的某个扇区号。为此，需有一个虚页号与辅存物理地址的映像表，该表称为外页表（相对应的前述页表称为内页表）。图 4-44 示出了外页表的结构。它也按虚页号的顺序排列，每个虚页在外页表中占有一项，记录该页在辅存中的物理地址。外页表的内容是在把程序装入辅存时填写的，其中 M 为装入位。

M	盘机号	柱面号	磁头号	扇区号
…	…			

图 4-44　外页表结构

通常外页表是存在辅存中的。当某个程序初始运行时，就把外页表的内容抄录到已建立的内页表的实页号字段中。在进行虚地址到实地址变换时，若出现页面失效，从内页表的实页号字段中取出的正是辅存物理地址。而当该页调入主存后，其实页号字段被真正填入所在主存的实际页号。

由于虚拟存储器的页面失效概率较低，一般不到1%。因此，由虚地址变换成辅存物理地址完全可由软件实现，而不必提供专用的硬件支持。页面失效时，由于访问辅存需经机械动作，速度较低，因而不是让处理机空等着该页由辅存调入主存，而是切换到其他已准备就绪的任务（进程），把调页工作交由I/O处理机即通道来完成。虽然，任务（进程）切换一般需要执行几百甚至几千条指令，但比起调页来说，耗费的时间毕竟要小得多。

虽然页表是实现分页式虚拟存储器的关键，但同时注意到，在页式虚拟存储器中，每次访存都要增加一个查页表的过程，而页表是存放在主存中的，这就相当于每次多增加了一次访主存的操作。如果这个查页表的操作时间不能缩短，虚拟存储器是没有实用价值的。因此，如何加快地址变换速度，就成为提高虚拟存储器速度的另一个关键问题，一种较好的解决办法是采用所谓"快表"的方法。

仔细分析实际查表的过程可以发现，由于程序局部性的特点，对页表内各页表项的使用不是随机的，而是簇聚的，即在某一段时间内，实际上只用到页表中很少的几个页表项。因此，可以单独采用快速硬件，实现只含有部分页表项的地址映像变换表，这种用快速硬件实现的部分页表称为转换旁视缓冲器（Translatim Lookaside Buffer，TLB），又称快表，而相对地把原来的页表称为慢表。快表比慢表小得多，其内容只是慢表的一个小的副本。

快表可用相联存储器实现，其中的各页表项含有最近访问的虚页号及其对应的主存实页号。图4-45给出了使用快表、慢表结合进行地址变换的过程。

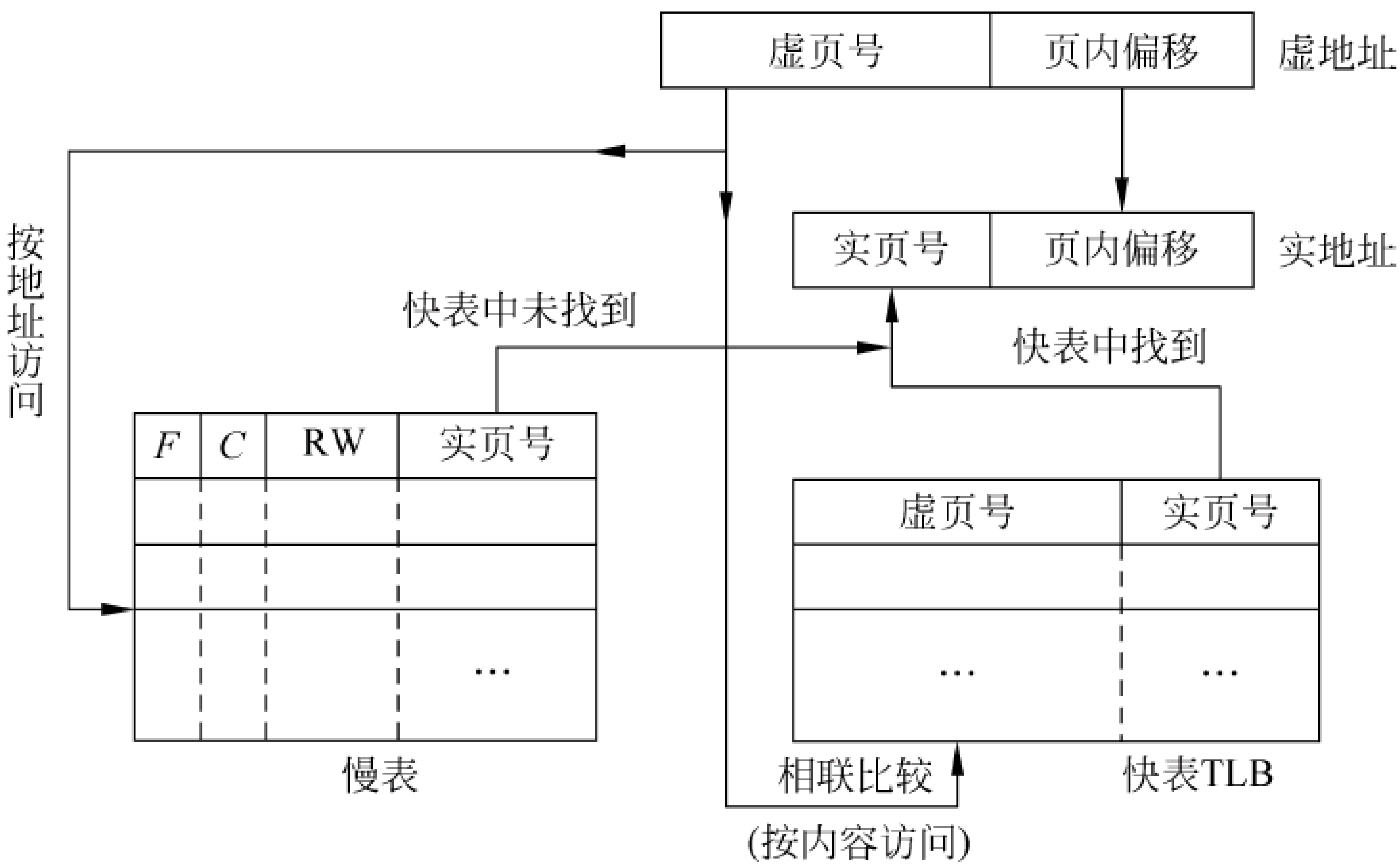


图4-45　快慢表结合实现虚地址变换

首先按虚地址中的虚页号同时到快表和慢表中进行查找。在快表中采用的是相联比较（按内容访问），若该虚页号已在快表中，则可以直接从对应的项中取出实页号，与虚地址中的页内偏移地址一道共同形成访主存的实地址，并同时使查慢表的操作作废。若该虚页号不在快表中，则在慢表中继续查找，一方面通过慢表继续完成虚地址的转换工作，另一方面

将找到的虚页号和对应的实页号同时存入快表中。

由上可见,快表和慢表又构成了一个表层次。如果快表的命中率相当高,则使地址变换所需时间接近于快表。

4.5.3 虚拟存储器的分段式管理

上述虚拟存储器的分页式管理控制简单,实现起来较为容易。然而它却存在两个方面的问题。

一方面当虚存空间很大时,页表也将非常庞大。例如,假设实存空间为256MB,虚存空间为4GB,页长为512B,则虚存共有8M个页,也就意味着页表共有8M个页表项。而每个页表项中需存储28位的实存地址和控制信息等,共需要至少4个字节。如果这样,页表需占用主存空间为8M×4B=32MB。如果虚存空间进一步增大,页表规模也将进一步增大,占用主存空间的容量也将更多。

另一方面虚拟存储器按页分配机制并不符合程序存储空间分配的特性。一般程序都采用模块化结构设计,这些模块可能是过程、子程序或函数调用等。从程序的自然特性讲,一个模块能够一次完整地调入主存,便于模块的运行管理和存储保护。然而分页式虚拟存储器却不能保证这一点。一个模块可能分布在不同的页中,这些页调入主存的时间可能是不同的,而且在主存空间分配上有可能是不连续的,这就使得今后系统对这一模块的存储管理和保护较为困难。

虚拟存储器的分段式管理就是按照程序的自然结构进行主存空间分配的,每次总是将一个程序模块完整地调入主存中。在这里,一个程序模块又称为一个段,为其分配的主存空间称为一个块。由于不同的程序模块长度是不同的,因此块长也是可变的。

为实现对长度不同的段进行空间分配,操作系统必须对主存空间进行有效的管理,例如,通过建立一些表格来了解主存空间的占用和可用情况。

(1) 主存占用空间表:它的每一项都指出所分配给的一个段名、块所占用区域的基地址和块的长度,如表4-3所示。

(2) 主存可用空间表:它的每一项指出一个未被占用区的地址和长度,如表4-4所示。

表4-3 主存占用空间表

段名	块基址	块长
…	…	…

表4-4 主存可用空间表

可用块基址	可用块长
…	…

当一个程序段需要从辅存调入主存时,系统首先在主存可用空间表中查找是否有能满足该段空间分配的主存块。若有,则为其分配,并在主存占用空间表中增加一项,填上所调入段的名字、该主存块的基址和块长,而将主存可用空间表中已分配的该块删除。当一个程序段不再需要留在主存时,则要将该段占用的主存块释放,同时在主存可用空间表中增加一项可用块。

对于长度不一致的块,有两种广泛使用的分配算法,即首次匹配和最佳匹配法。首次匹配算法是指对主存可用空间表从头开始一次查找,直到找到一个能满足一个程序段空间要

求的主存块。最佳匹配算法是指每次都要查询整个可用空间表，选择一个能满足程序段空间要求的最小主存块进行分配。现举例说明这两种算法的区别。

假设某个时候主存的空间初始状态如图 4-46(a)所示。其中已有三个程序段 K1、K2 和 K3，阴影部分为可用的空白区，每个存储区的左边表示该块的大小。

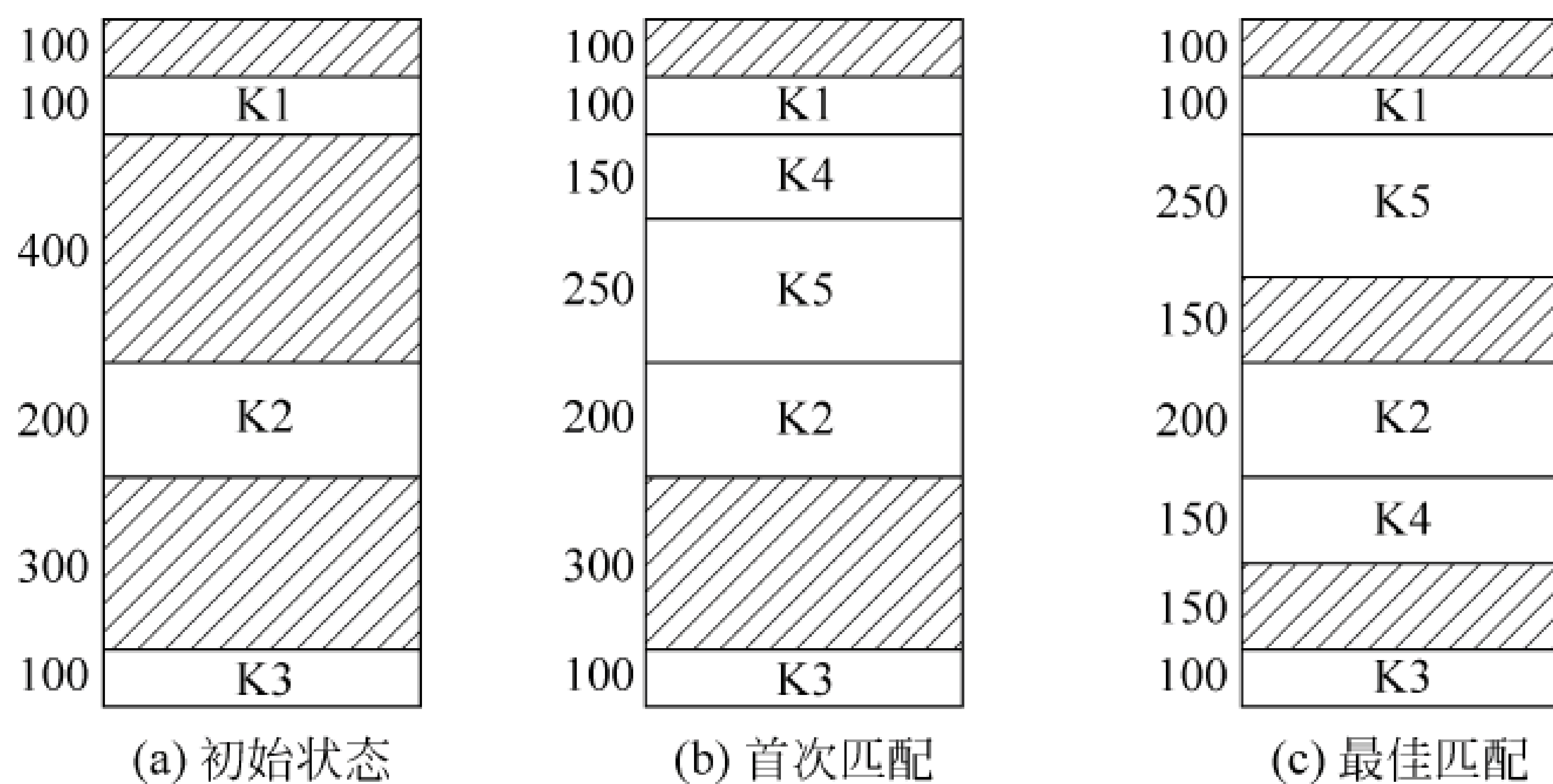


图 4-46 段式存储分配

假定现在要使长度分别为 150 与 250 的块 K4 和 K5 分配到主存。图 4-46(b)和图 4-46(c)分别给出了采用首次匹配与最佳匹配算法所得的结果。这两种算法都已经实现并取得了令人满意的结果。比较起来，首次匹配算法所需执行时间比最佳匹配算法要小。这是因为，首次匹配算法无须像最佳匹配算法那样每次都要查找整个区域。另外，由于首次匹配法具有把所有块集中分配到存储器某一端的趋向，那么可用空间的大区域可能在存储器的另一端出现，可用来容纳今后大的块的调入，也使得存储空间不太容易像最佳匹配法那样出现碎片的现象。例如，在 K4 和 K5 分配完的基础上，现需再分配一个大小为 200 的块，采用首次匹配法的图 4-46(b)就可以实现，而采用最佳匹配法的图 4-46(c)则无法实现，因为再也找不到一个 200 以上的单独的块了。因此，一般来讲，首次匹配法更适合于段式存储空间的分配。

在分段式虚拟存储器中，虚地址也分为两个部分：段号和段内地址。为实现虚地址到实地址的变换，系统为此专门建立了一个段表。段表由一个个段表项组成，每一个段表项对应调入主存的一个段，段表项的主要内容包括段在主存的起始地址、段长、装入位 F 和访问控制位 RW。段表存放在主存中，其在主存中的起始地址由一个段基址寄存器指出。段表的长度是不固定的，并随着用户程序段的调进调出而变化。当一个程序段被调入主存，就会在段表中登记一项，记录该段在主存中所分配的存储空间的起始地址等信息。当进行地址变换时，首先根据虚地址中的段号在段表中找到该段对应的段表项。若 $F=1$，则访问有效，于是从段表项中取出段首地址，与虚地址中的段内地址共同形成一个访存的实地址。若 $F=0$，则访问失效，需从辅存中将该段调入主存。分段式虚拟存储器的地址变换，如图 4-47 所示。

段式虚拟存储器的优点是：按段分配存储空间，一方面符合程序模块化的特点，另一方面便于程序段在主存中的管理和保护。后者是通过段表项中的段长来实现的。当进行地址变换时，需将虚地址中的段内地址与段表项中的段长进行比较，若段内地址大于段长，则说明该虚地址的访问超出了该段的界限，于是系统会产生一个越界错误，禁止本次访问。段式虚拟存储器的缺点是：当系统进行了频繁的块调进和调出后，会使主存出现碎块现象，导致后续的段无法分配的情况出现。

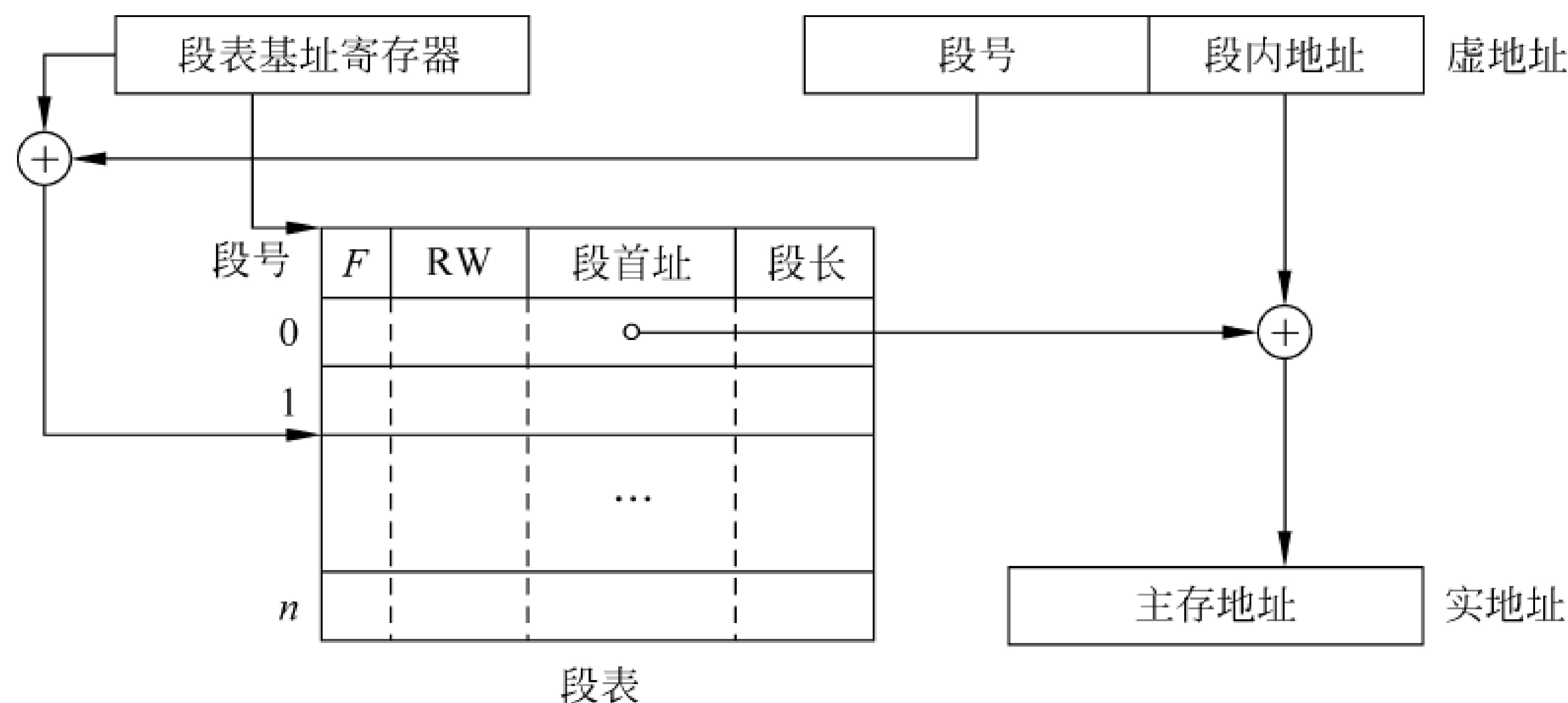


图 4-47 分段式虚拟存储器地址变换

4.5.4 虚拟存储器的段页式管理

上述的分页式管理与分段式管理结合起来就形成了段页式虚拟存储器系统，它可以兼得两者的优点，即一方面具有分页式简单、不会产生碎片等的优点，另一方面又具有按程序自然段进行空间分配所带来的存储管理和存储保护等方面的优点。在段页式系统中，程序首先按模块分段，然后再对每个段划分固定大小的页。这就使得程序段在主存的调进调出可以按页进行空间分配，同时又可以按段进行管理和保护。

为实现段页式管理，系统为每个调入主存运行的程序单独建立一个段表和一组页表。段表的每一项对应一个段，其内容包括该段对应的页表的起始地址及相关控制信息。段表在主存中的起始地址由一个段表基地址寄存器指出。页表用于指明该段各页在主存的分配情况，其内容包括在主存的页号、装入位 F 和修改位 C 等。在段页式虚拟存储器中，用户程序的地址由三部分组成：段号、页号和页内偏移。地址变换过程如图 4-48 所示。

首先根据段表基地址寄存器的内容和虚地址中的段号找到该程序段的段表项，然后根据段表项中的页表首址和虚地址中的页号找到对应的页表项，最后从页表项中取出实页号，与虚地址中的页内偏移一起形成访主存的实地址。

可以看出，段页式虚拟存储器系统在进行虚地址到实地址变换时至少需要经历两次查表过程，即先查段表，再查页表。因此，占用的时间也就更多。而且，当一个页表的大小超过一个页面的大小时，页表就被分成了几个页，这时又需设置二级页表，构成二级页表层次。一个大的程序可能需要设置多级页表层次。对于多级页表层次，在程序运行时，除了第一级页表需驻留在主存之外，其他页表可存放在辅存中，需要时再由第一级页表调入，从而减少每道程序页表所占主存的空间。

为加快查表的速度，在段页式虚拟存储器中同样需要采用快慢表结合的方式进行虚地址的变换。

在多用户多道程序系统中，允许有多个用户的程序同时运行，这时需要使用一个基号（又称用户标识号）来标识和区分所运行的每道程序。系统会为每道用户程序设置一个段表基址寄存器，用于指明系统为用户程序建立的段表在内存中的起始地址。而虚地址中需增加一个基号字段，如图 4-49 所示。

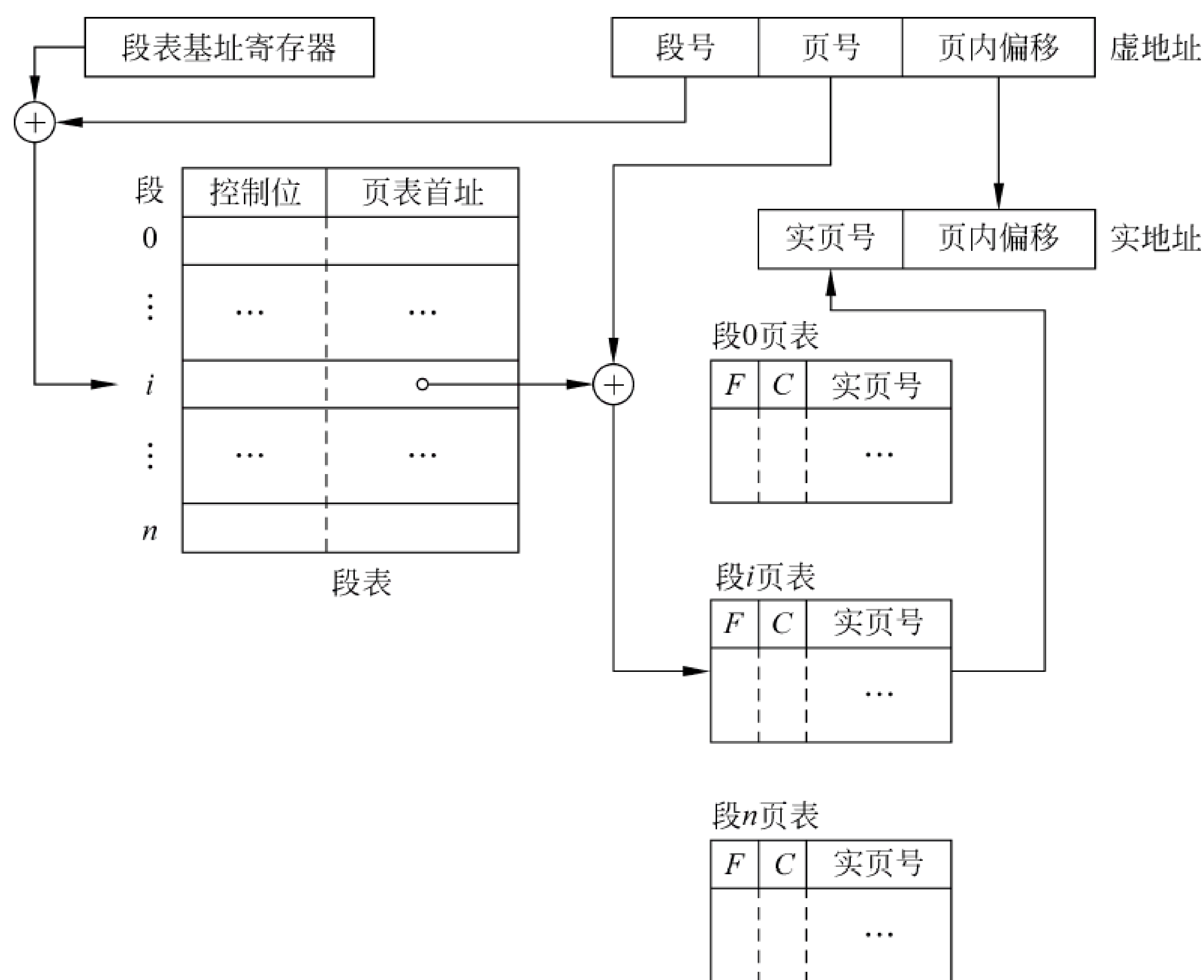


图 4-48 段页式虚拟存储器地址变换

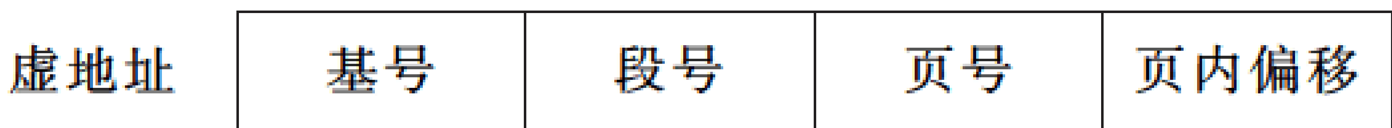


图 4-49 在虚地址中增加一个基号字段

【例 4.4】 假设某时间机器中运行有三道程序(用户标识号分别为 A、B、C),它们对应的段表基址寄存器内容分别为 S_A、S_B 和 S_C。现假设程序中给出了一个虚地址 C-1-2-0100(即程序 C 的第 1 段第 2 页中的偏移地址为 0100 的单元),试通过图示的方式说明其虚地址到实地址的变换过程。

解:本例中,设 A、B、C 三道程序分别有 3、2、3 个段,其中程序 C 的三个段分别为 C_0、C_1、C_2,各自对应内存中的一个页表,三个段长度分别为 2、3、2 页。将虚地址 C-1-2-0100 变换为实地址的过程如图 4-50 所示。

首先根据虚地址中的基号找到对应程序 C 的基址寄存器,按基址寄存器中的地址找到该程序在内存中的段表;紧接着由虚地址中的段号找到段表中的相应段 1 项,按段 1 项中的页表首址找到段 1 对应的页表;再由虚地址中的页号找到页表中的页表项 2,从此项中取出该虚页在内存中的实页号 10;最后由实页号 10 和页内地址 0100 一起共同形成访主存的实地址。

4.5.5 虚拟存储器的替换策略

在虚拟存储器中,由于虚存空间比实存空间大得多,必然会出现主存中所有页已经全部被占用的情况。这时如果又有新的页需要从辅存调入主存时就会发生页面失效。当发生页面失效时,就必须将主存中的某个页替换出去,以接纳来自辅存要调入的页。那么,按照什

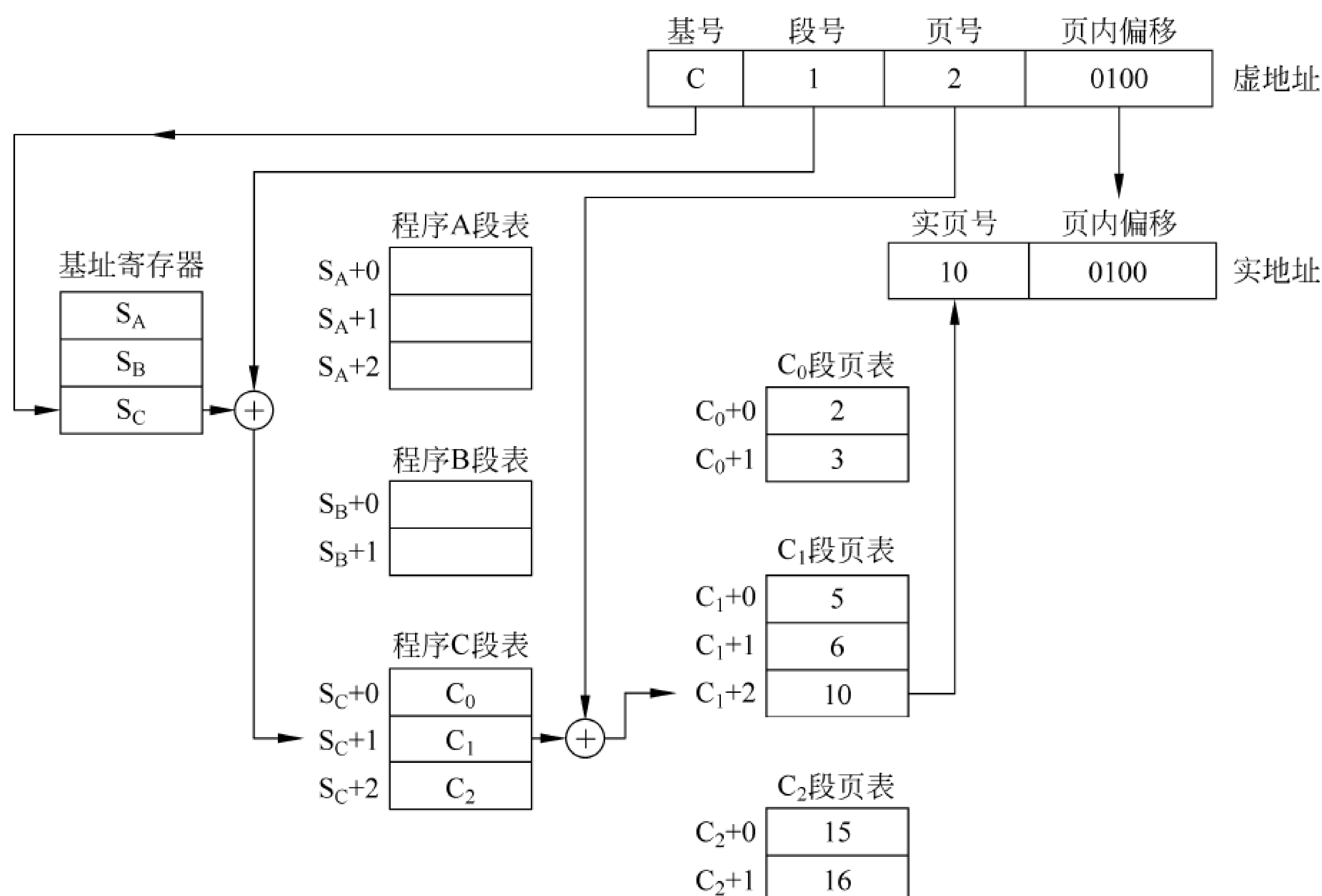


图 4-50 段页式虚地址转换

么规则替换主存中的页呢？这就是虚拟存储器替换算法要解决的问题。

选择替换算法，一方面要看这种算法是否反映了程序的局部性，使主存的命中率得以提高；另一方面要看这种替换算法是否易于实现。虚拟存储器中的页面替换策略和 cache 中的块替换策略有很多相似之处，但有三点显著不同：

（1）当虚拟存储器发生页面失效时至少要涉及一次对辅存的存取操作，而对辅存的存取操作速度远比对主存的读取操作慢，从而使系统所遭受的时间上的损失要远比 cache 未命中时大得多。

（2）虚拟存储器的页面替换是由操作系统软件来完成的。

（3）由于主存空间远比 cache 空间大，主存与虚存空间之间又是全映射方式，因此虚拟存储器页面替换的选择余地很大，属于一个进程的页面都可替换。

虚拟存储器中的替换策略一般采用 LRU 算法、LFU 算法 FIFO 算法，或将两种算法结合起来使用。

对于将被替换出去的页面，假如该页调入主存后没有被修改过，就不必进行处理，否则就必须把该页重新写入辅存，以保证辅存中数据的正确性。

【例 4.5】 假设主存只有三个页面，某程序访问虚页面的序列是 2、3、2、1、5、2、4、5、3、2、5、2 页。试求分别采用 FIFO 和 LRU 替换算法时的命中率。

解：分别采用 FIFO 和 LRU 替换算法时的主存页面变化情况如图 4-51 所示。

其中 FIFO 算法在 12 次访问中命中 3 次，命中率为 25%；而 LRU 算法命中 5 次，命中率为 42%。

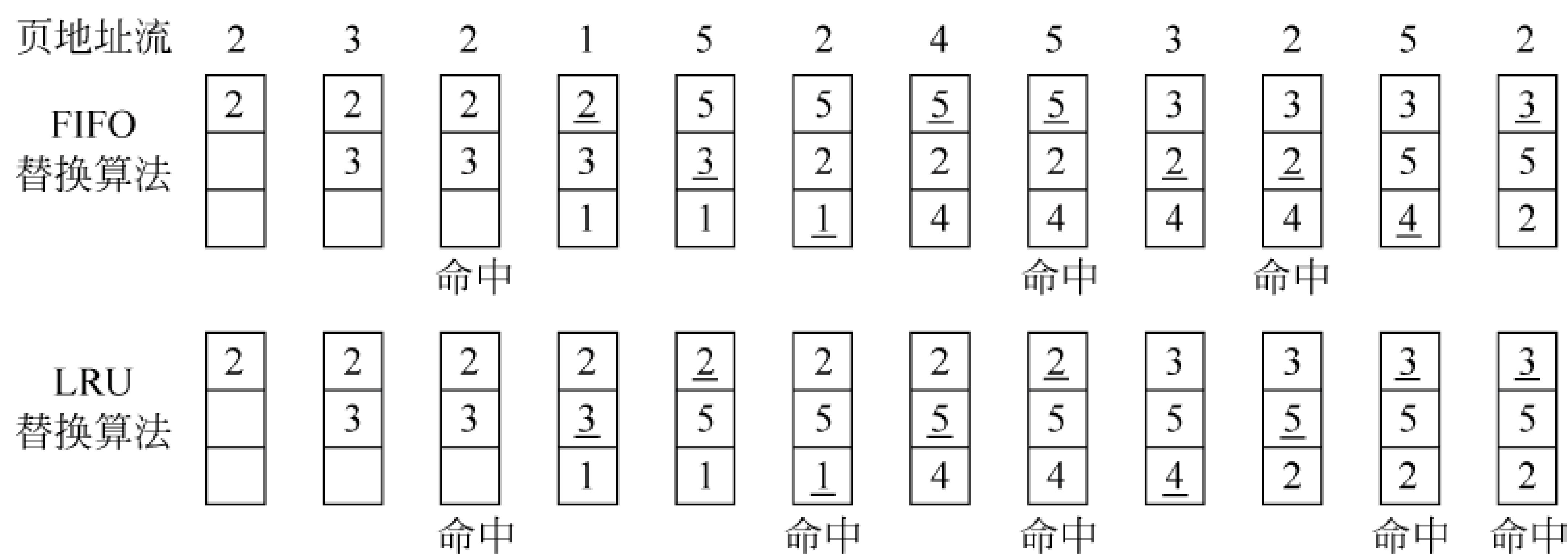


图 4-51　主存页面变化情况

4.5.6　虚拟存储器举例：Pentium 的虚拟存储器组织

1. Pentium CPU 的虚地址模式

Pentium CPU 的虚拟存储器管理是通过存储管理部件 MMU 来完成的，MMU 包括分段部件(SU)和分页部件(PU)两部分，它允许 SU 和 PU 单独工作或同时工作，从而支持以下三种受保护的虚拟地址模式(简称保护模式)：

1) 分段不分页模式

虚拟地址(在 Pentium CPU 中称为逻辑地址)由一个 16 位的段选择符和一个 32 位的偏移地址组成。段选择符的高 14 位用于指定具体的段，最低 2 位用于段保护。一个进程可拥有的最大虚拟地址空间为 $2^{14+32}=2^{46}=64$TB。分段部件(SU)将二维的分段虚拟地址转换成一维的 32 位线性地址。

这种模式的优点是无须访问页目录和页表，地址转换速度快。另外，对段提供的一些保护定义可以一直贯通到段的单个字节级。

2) 不分段分页模式

在这种模式下 SU 不工作，只是分页部件(PU)工作。程序也不提供段索引，程序中使用的寄存器提供的 32 位地址被看成由页目录、页表、页内偏移三个字段组成。由 PU 完成虚拟地址到物理地址的转换。进程可拥有的最大虚拟地址空间为 $2^{32}=4$GB。

这种分页模式虽然减少了虚拟空间容量，但一方面也能提供保护机制，另一方面比分段模式具有更大的灵活性。

3) 分段分页模式

这是一种在分段基础上增加分页存储管理的模式。即将 SU 部件转换后的 32 位线性地址看成由页目录、页表、页内偏移三个字段组成，再由 PU 部件完成两级页表的查找，将其转换成 32 位物理地址。一个进程可拥有的最大虚拟地址空间也为 64TB。这种模式兼顾了分段模式的存储保护和分页模式灵活的优点。

2. 保护模式下的分段管理

在分段模式下，每一个程序都被分成若干个段，且一个程序至少包含一个代码段和一个数据段。每一个程序段都有自己的基地址、段界限以及保护信息，这些信息被保存在由操作系统管理的全局描述符表(Global Descriptor Table，GDT)或局部描述符表(Local

Descriptor Table,LDT)内,这两个表的每一项称为段描述符,由8个字节组成,也就是说,段描述符中存储了程序段的基地址、段界限以及保护信息等。

分段模式下的虚拟地址(或逻辑地址)包含一个16位的段选择符和一个32位的偏移地址,其中段选择符的格式如图4-52所示。

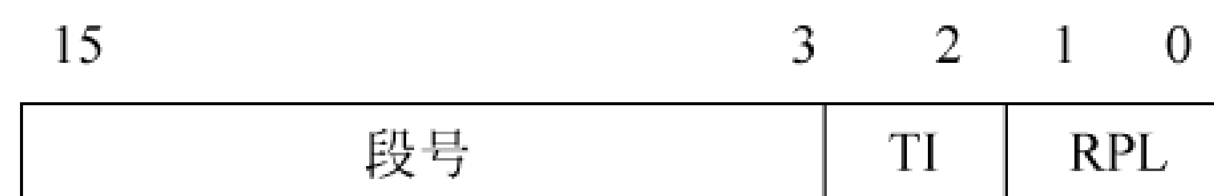


图 4-52 段选择符的格式

段选择符中包含了下列域。

(1) 段表指示符(TI):表示对应的程序段是在GDT中还是在LDT中。

(2) 段号:对应GDT或LDT中的一项。

(3) 请求特权级(RPL):表示这个请求的特权级。

分段模式下由逻辑地址转换成32位物理地址的示意图,如图4-53所示。

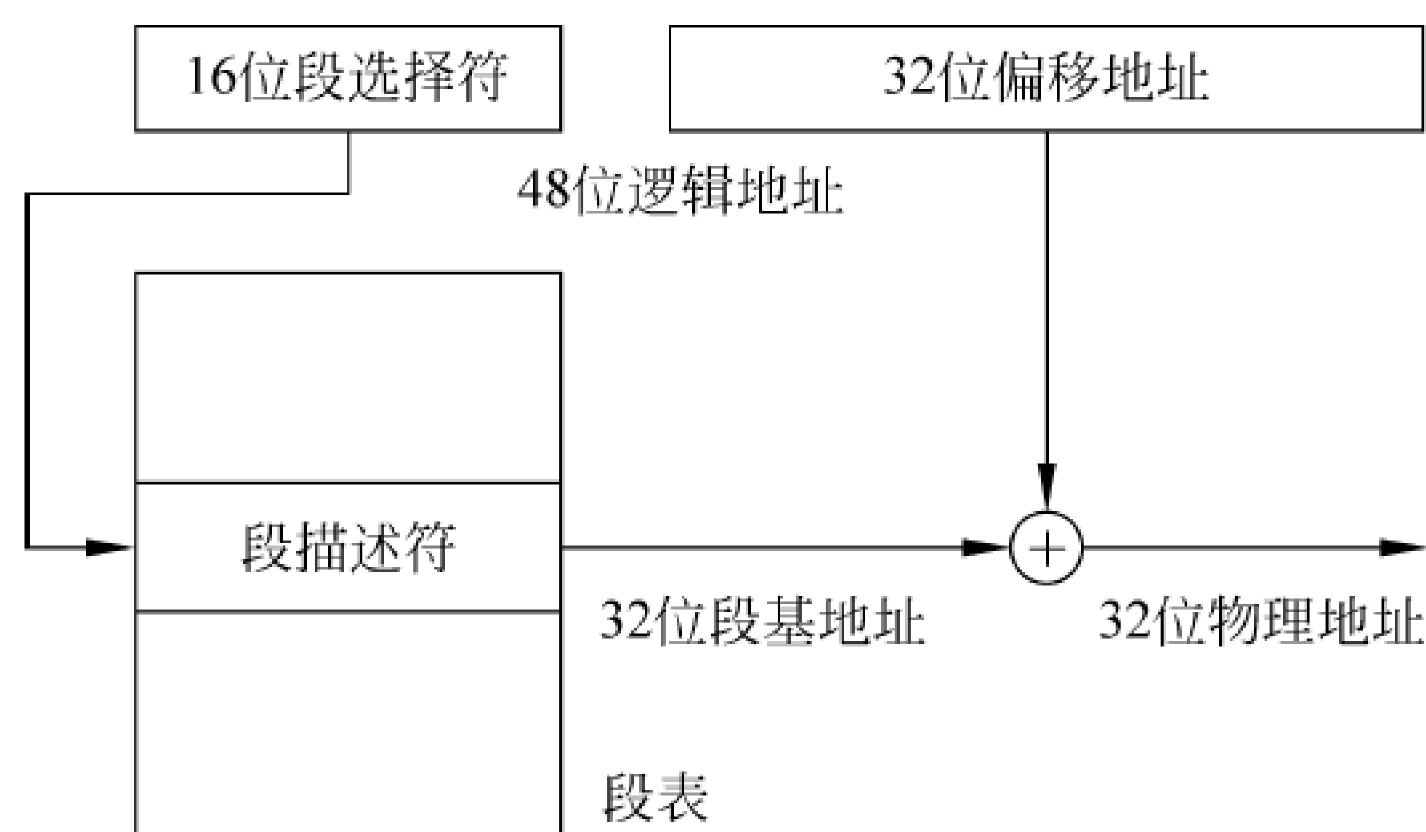


图 4-53 分段模式下地址转换

3. 保护模式下的分页管理

Pentium CPU有两种分页管理方式:一种是将页划分成4KB大小,使用二级页表(页目录表和页表)进行地址转换;另一种是将页划分成4MB大小,使用单级页表进行地址转换。这里仅介绍后一种方式。

单级页表由一个个页表项组成,页表项的格式如图4-54所示。

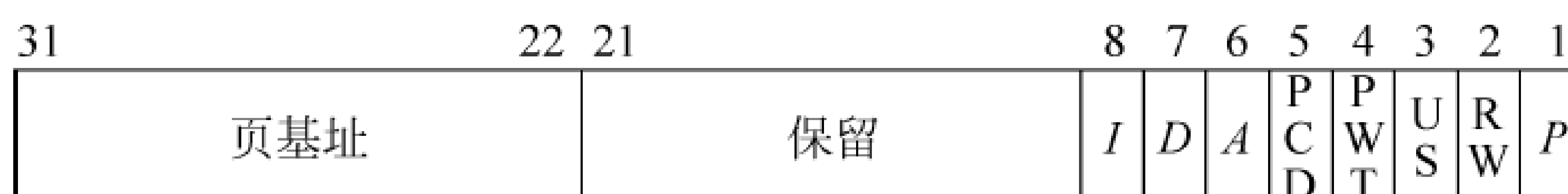


图 4-54 页表项的格式

其中,页基址10位,其他各位的含义如下。

(1) I 位:指示页面大小,$I=1$,表示页面大小为4MB;$I=0$,表示页面大小为4KB。

(2) P 位:存在位,$P=1$,表示该页已装入主存,$P=0$ 时访问该页将产生缺页错误。

(3) D 位:修改位,$D=1$,表示该页被修改过。

(4) A 位:访问位,该页装入主存后若被访问过,则置 $A=1$,否则 $A=0$。

(5) PCD 位：用于页 cache 的禁止控制。

(6) PWT 位：用于全写法的控制。

(7) US 位：用于用户/监督控制。

(8) RW 位：用于读/写控制。

分页方式的地址转换过程如图 4-55 所示。

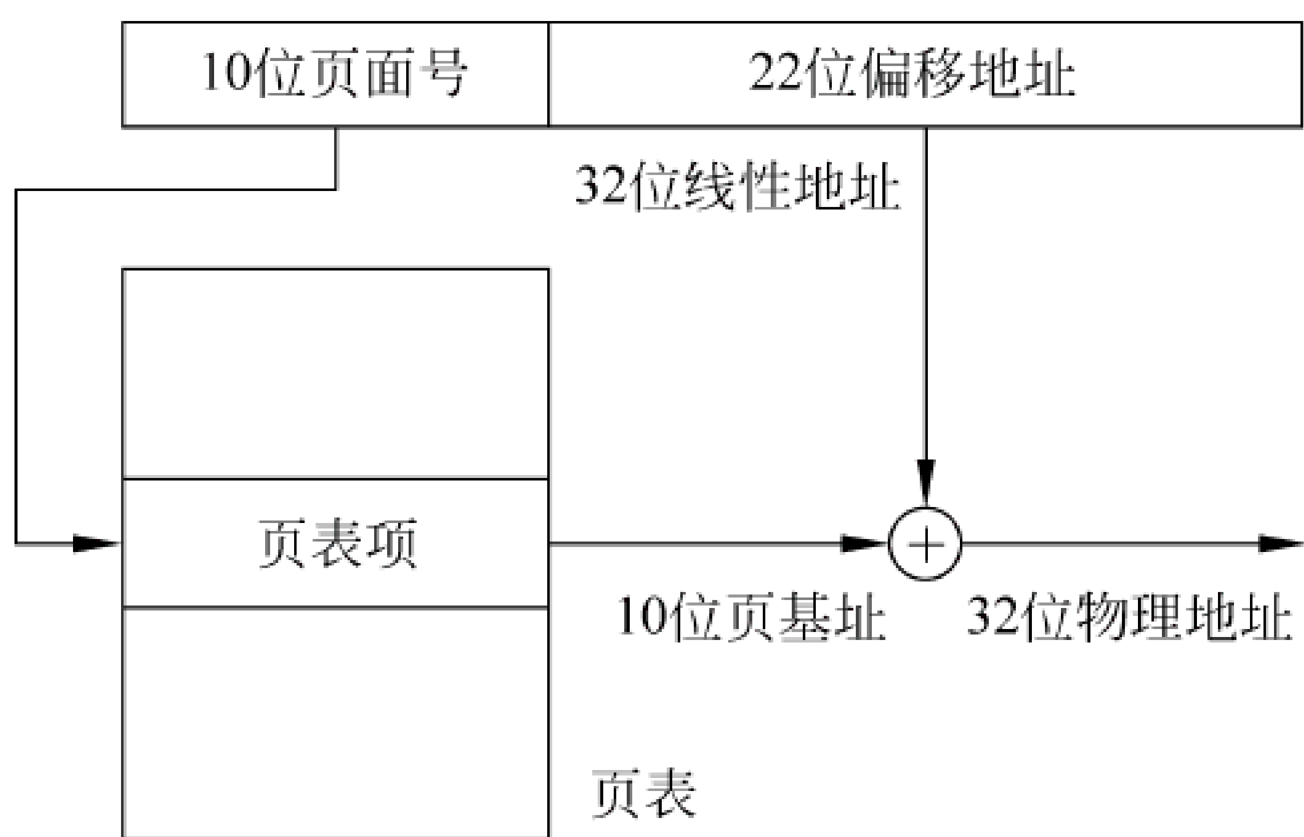


图 4-55　分页模式下地址转换

本章小结

本章主要讲述了以下内容：

(1) 存储器的组织。存储器在物理上是由一定的具有存储功能的器件构成的，在逻辑上可以看成由一个个存储单元构成的线性地址空间。

(2) 存储器的分类。从功能上存储器分为主存和辅存，主存主要由半导体存储器件构成，而辅存则由磁介质存储器或光存储器构成。

(3) 计算机主存储器的种类、单元电路的构成及工作原理和主存储器的组成及设计，还讲述了提高访存效率的交叉存储技术等。

(4) 现代存储器系统的存储层次，包括 cache-主存层次和主存-辅存层次，讲述了它们所基于的程序局部性原理、各层次的工作原理和实现方法等。

习题四

一、名词解释

1. RAM　　2. ROM　　3. 存储周期　　4. 存储层次

5. 程序的局部性原理　　6. cache　　7. 虚拟存储器　　8. LRU

二、选择题

1. 存储器是计算机系统中的记忆设备，它主要用来(　　)。

A. 存放数据　　B. 存放程序

C. 存放数据和程序　　D. 存放微程序

2. 存储单元是指(　　)。

A. 存放一个二进制信息位的存储元　　B. 存放一个机器字的所有存储元集合

C. 存放一个字节的所有存储元集合　　D. 存放两个字节的所有存储元集合

3. 和外存相比,内存的特点是(　　)。

A. 容量大、速度快、成本低　　B. 容量大、速度慢、成本高

C. 容量小、速度快、成本高　　D. 容量小、速度慢、成本低

4. 某 SRAM 芯片的存储容量为 256K×4 位,则该芯片的地址线和数据线分别为(　　)。

A. 18,4　　B. 4,18　　C. 20,8　　D. 8,20

5. 某机器字长为 32 位,其存储容量为 1MB,若按字编址,它的寻址范围为(　　)。

A. 1M　　B. 512KB　　C. 256K　　D. 256KB

6. 下列因素中,与 cache 的命中率无关的是(　　)。

A. 主存的存取时间　　B. cache 块的大小

C. cache 的组织方式　　D. cache 的容量

7. 下列说法中正确的是(　　)。

A. 多体交叉存储器主要解决扩充存储容量问题

B. cache 与主存统一编址,cache 的地址空间是主存地址空间的一部分

C. cache 的功能全部由硬件实现

D. 虚拟存储器的功能全部由硬件实现

8. 某容量为 256MB 的存储器,由若干 4M×8 位的 DRAM 芯片构成,该 DRAM 芯片的地址引脚和数据引脚总数是(　　)。(2014 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

A. 19　　B. 22　　C. 30　　D. 36

9. 采用指令 cache 与数据 cache 分离的主要目的是(　　)。(2014 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

A. 减低 cache 的缺失损失　　B. 提高 cache 的命中率

C. 减低 CPU 平均访问时间　　D. 减少指令流水线资源冲突

10. 某计算机主存地址空间大小为 256MB,按字节编址。虚拟地址空间大小为 4GB,采用页式存储管理,页面大小为 4KB,TLB(快表)采用全相联映射,有 4 个页表项,内容如下表所示。则对虚拟地址 03FF F180H 进行虚实地址变换的结果是(　　)。(2013 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

A. 015 3180H　　B. 003 5180H　　C. TLB 缺失　　D. 缺页

有效位	标记	页框号	…
0	FF180H	0002H	…
1	3FFF1H	0035H	…
0	02FF3H	0351H	…
1	03FFFH	0153H	…

11. 下列关于闪存(Flash Memory)的叙述中,错误的是(　　)。(2012 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

A. 信息可读可写,并且读、写速度一样快

B. 存储元由 MOS 管组成,是一种半导体存储器

C. 掉电后信息不丢失,是一种非易失性存储器

D. 采用随机访问方式,可替代计算机外部存储器

12. 假设某计算机按字编址,cache 有 4 个行,cache 和主存之间交换的块为 1 个字。若 cache 的内容初始为空,采用 2 路组相联映射方式和 LRU 替换算法。当访问的主存地址依次为 0,4,8,2,0,6,8,6,4,8 时,命中 cache 的次数是( )。(2012 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

A. 1　　B. 2　　C. 3　　D. 4

三、综合题

1. 简述存储器的物理特性和逻辑特性。

2. 存储器与 CPU 之间主要通过哪几种信号连接? CPU 对存储器的访问主要有哪几种?

3. 说明 CPU 对存储器进行读和写操作的过程。

4. 计算机存储器都有哪些种类?

5. 计算机主存中为什么需要包含小容量的 ROM?

6. 现代计算机存储器为何采取层次结构? 有哪两个主要层次? 分别解决什么问题?

7. 静态 RAM 和动态 RAM 存储器从单元电路构成上来讲最主要的不同在哪里? 这种不同对这两种存储器的使用有什么影响?

8. 某用户在进行一个小型控制电路板设计时需要用到一个小容量的 ROM 芯片,在实验阶段,他需要少量的 ROM 芯片做实验用,实验成功后,他需要大量 ROM 芯片用在控制电路板上进行商业销售。请问他在实验阶段和商业销售阶段分别使用什么样的 ROM 芯片比较好?

9. 有一个 1M×32 位的存储器,由 128K×8 位的 DRAM 芯片(芯片的内部单元组成一个 1024×1024 的阵列)构成,试问:

(1) 需要多少个这种芯片?

(2) 画出此存储器的连接图。

(3) 采用逐行刷新方式,且要求单元刷新间隔不超过 8ms,则刷新周期是多少(即每间隔多长时间刷新一行)?

10. 设 CPU 有 16 根地址线,8 根数据线,并用 MREQ 作访存控制信号(低电平有效),用 WR 作读/写控制信号(高电平为读,低电平为写)。现有下列存储芯片: 1K×4 位 RAM; 4K×8 位 RAM; 8K×8 位 RAM; 2K×8 位 ROM; 4K×8 位 ROM; 8K×8 位 ROM 及 74LS138 译码器和各种门电路,画出 CPU 与存储器的连接图,要求:

(1) 主存地址空间分配:

6000H～67FFH 为系统程序区;

6800H～6BFFH 为用户程序区。

(2) 合理选用上述存储芯片,说明各选几片?

(3) 详细画出存储芯片的片选逻辑图。

11. 设某 8b 的机器其地址总线为 20b,其 1MB 的存储器由 RAM 和 ROM 两部分组成,其中 ROM 占 128KB(使用 64K×8 的芯片),剩余的为 RAM 空间(使用 128K×4b 的芯片),试问:

(1) ROM 和 RAM 芯片各分别需用多少片?

(2) 试画出 ROM 和 RAM 的容量扩展图。

(3) 设 RAM 芯片的控制信号为片选$\overline{CS}$和读写$\overline{WE}$,ROM 芯片的控制信号为片选$\overline{CS}$和输出允许$\overline{OE}$,CPU 对存储器芯片的访问控制信号为 R/W 和 M/IO,试画出 CPU 与 RAM 及 ROM 的连接图。

12. 某机器采用四体交叉存储器,今执行一段循环程序,此程序放在存储器的连续地址单元中。假设每条指令的执行时间相等,而且不需要到存储器中存取数据。请问在下面两种情况中,程序的运行时间是否相同?

(1) 循环程序由 6 条指令组成,重复执行 80 次。

(2) 循环程序由 8 条指令组成,重复执行 60 次。

13. 设某计算机主存容量为 64MB,cache 容量为 256KB,主存和 cache 的数据块大小为 32B,则主存地址和 cache 地址各为多少位? 主存和 cache 之间交换的数据单位是什么? 在采取全相联映射方式的情况下,主存地址和 cache 地址格式分别是怎样?

14. 在一个使用 cache 的机器中,CPU 执行某一段程序共完成 3000 次访存操作,其中命中 cache 的次数为 2420 次。已知 cache 的存储周期为 10ns,主存存储周期为 80ns,求 cache-主存系统的效率和 CPU 访问该存储系统的平均访问时间。

15. 设某机器的 cache 采用 4 路组相联映射方式,已知 cache 容量为 16KB,主存容量为 2Mb,每个块有 8 个字,每个字为 32b。试问:

(1) 主存地址是多少位(按字节编址)? 地址格式是怎样的?

(2) 设 cache 起始为空,CPU 从主存单元 0、1、…、100 依次读出 101 个字(一次读出一个字),并重复操作 11 次,问 cache 的命中率为多少? 若 cache 速度是主存的 5 倍,则采用 cache 和不采用 cache 速度可提高多少倍?

16. 已知某机器主存容量为 256MB,虚存容量为 4GB,试问虚地址和实地址各为多少位? 如采用分页式虚拟存储器管理,每个页面为 4KB,则页表大小是多少?

17. 假设某机器中运行的一个进程访问的页面序列是 0、1、2、4、2、3、0、2、1、3、2,针对以下两种情况分别画出采用 FIFO 和 LRU 替换算法时的主存页面变化情况。

(1) 主存有 3 个页面。

(2) 主存有 4 个页面。

18. 某计算机有一个 cache、主存和用于虚拟存储器的磁盘。如果有一个字在 cache 中,则需 20ns 的时间来存取它;如果字在主存而不在 cache 中,则首先需要 60ns 的时间把它调入 cache,然后再开始引用;如果字不在主存,则需 12ms 的时间从磁盘中获取,再用 60ns 把它存入 cache。设 cache 的命中率为 0.9,主存的命中率为 0.6。试问:存取一个字的平均存取时间是多少?

19. 某计算机主存按字节编址,逻辑地址和物理地址都是 32 位,页表项大小为 4 字节。请回答下列问题。(2013 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

(1) 若使用一级页表的分页存储管理方式,逻辑地址结构为

页号(20 位)	页内偏移量(12 位)

则页的大小是多少字节？页表最大占用多少字节？

（2）若使用二级页表的分页存储管理方式，逻辑地址结构为

页目录号(10 位)	页表索引(10 位)	页内偏移量(12 位)

设逻辑地址为 LA，请分别给出其对应的页目录号和页表索引的表达式。

（3）采用(1)中的分页存储管理方式，一个代码段起始逻辑地址为 00008000H，其长度为 8KB，被装载到从物理地址 00900000H 开始的连续主存空间中。页表从主存 00200000H 开始的物理地址处连续存放，如图 4-56 所示(地址大小自下向上递增)。请计算出该代码段对应的两个页表项的物理地址、这两个页表项中的页框号以及代码页面 2 的起始物理地址。

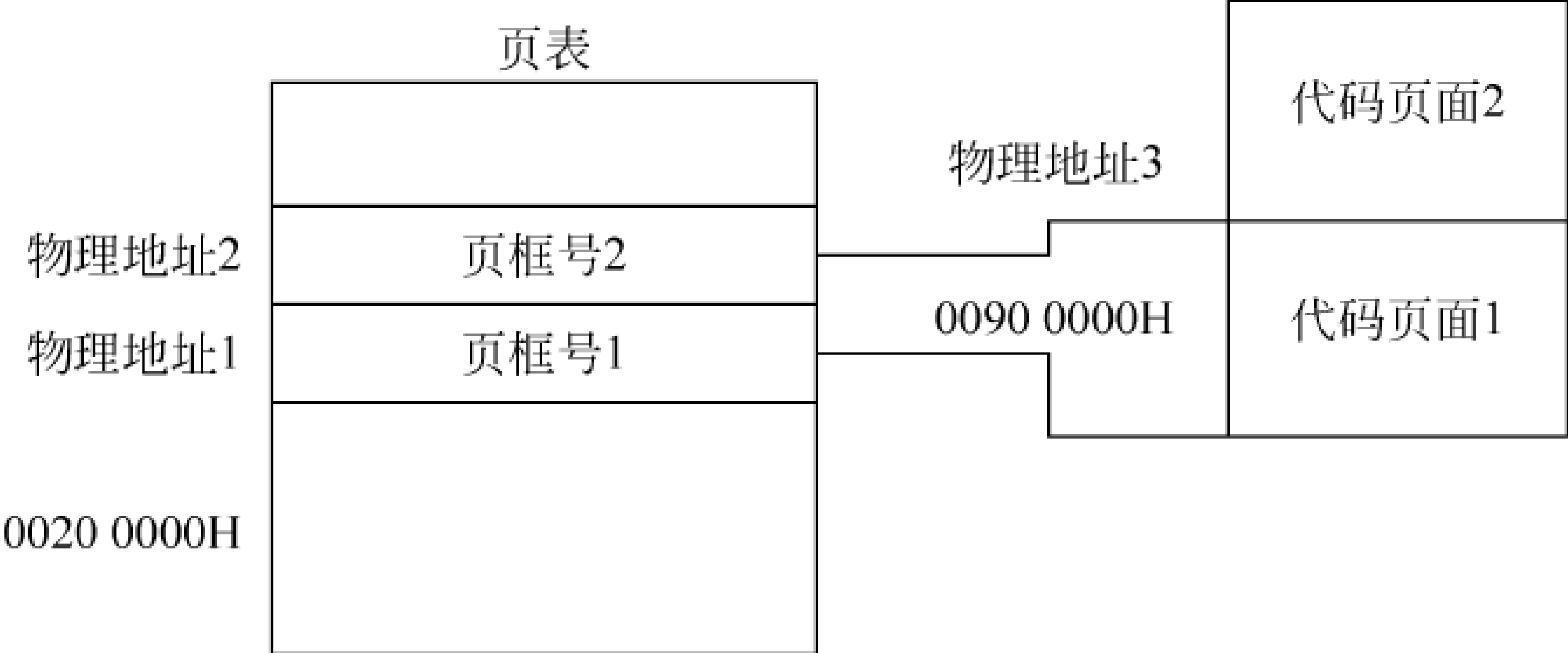


图 4-56　页表

第5章 输入输出系统组织

输入输出系统是计算机重要的组成部分之一，简单地说，输入输出系统的主要功能是完成程序和数据的输入及机器运行结果的输出。随着计算机应用的发展，计算机所能处理的数据种类越来越多，从简单的数值数据、文本数据到图形图像数据及音视频数据等各种多媒体数据，这对现代计算机的输入输出系统提出了更高的要求。

本章首先介绍计算机输入输出系统的组成，然后对计算机输入输出的控制方式进行详细讨论，最后介绍计算机存储设备——磁盘系统以及由磁盘阵列组成的 RAID 技术。

5.1 输入输出系统概述

输入输出系统主要由输入输出设备和与 CPU 相连接的输入输出接口逻辑组成，CPU 通过接口逻辑实现与输入输出设备之间的数据传输。

5.1.1 输入输出设备

输入输出设备又称外围设备(Periphery)，它涵盖的面非常广。事实上，除了 CPU 和主存外，计算机系统的其他部件都可看成外围设备。

外围设备种类繁多，随着计算机应用的发展，不断有新的外围设备推向市场。一般来讲，计算机的外围设备可以分为以下 5 大类。

1. 输入设备

输入设备是指完成人机数据输入的外围设备。人们设计、建造和使用计算机的目的是让计算机帮助人们更高效地完成一定的功能。早期的计算机最主要的应用是科学计算，现代计算机的应用则深入到人们生活的方方面面。不管人们要求计算机完成什么样的功能，首先需要将所编制的程序和处理的数据输入计算机，而程序和数据等的输入就需要输入设备来完成。早期的计算机是使用纸带读入机等来完成程序和数据的输入的，键盘和鼠标是目前计算机标准配置的输入设备。随着现代计算机在多媒体信息领域的应用，不断涌现出各种用于多媒体信息输入的设备，如扫描仪、游戏控制杆、语音输入设备、图像输入设备以及各种音视频输入设备等。

2. 输出设备

输出设备是指完成人机数据输出的外围设备。早期用于科学计算的机器需要通过输出

设备将程序运行的结果以某种方式输出给程序员，如通过显示器将结果显示在显示屏上，或通过打印机将结果打印在打印纸上等。而现代计算机的输出设备种类越来越丰富，如用于绘制工程图纸的绘图仪、MPC 的音箱设备以及各种音视频输出设备等。

3. 存储设备

在第 4 章中讲到，现代计算机存储器采取层次结构实现，即由“cache-主存-辅存”层次组成。按照现代计算机辅存所采用的存储介质分，主要包括磁介质存储器（磁盘存储器和磁带存储器）和光盘存储器。可执行程序在运行前首先通过输入设备输入，并以文件的形式存放在辅存中。另外，计算机处理的各种信息（数据、文本、图形、图像、声音、视频、动画等）也均以文件的形式存放在辅存中。

由磁盘存储器、磁带存储器或光盘存储器构成的辅存就是计算机的存储设备，在机器中，它们是作为设备由操作系统进行管理的。有关磁盘存储器、磁带存储器或光盘存储器的内容将在 5.3 节详细讲述。

4. 数据通信设备

计算机与计算机之间的数据通信有多种方式，如早期的通过模拟电话线实现的计算机之间点到点的通信，通信双方要使用数据通信设备——调制解调器完成数字信号与模拟信号的转换。现代计算机主要通过网络进行数据通信，每台计算机都是通过使用数据通信设备——网络适配器（又称网卡）连入网络的。正是借助各种数据通信设备，才实现了计算机之间的互联。

5. 过程控制设备

计算机的一个重要应用领域就是工业控制。工业控制中通常需要使用一些过程控制设备完成对某一工业过程的控制。与上述列举的输入输出设备不同的是，这些过程控制设备往往是针对一特定应用专门设计的，属于计算机系统的非标准外围设备。

5.1.2　输入输出接口

从 5.1.1 节的讨论看到，计算机外围设备种类繁多，而且互相之间差异很大，这些差异主要体现在以下几个方面。

1）物理特性方面

主要是指设备连接的方式和读写驱动方式等，如连接口的类型、机械尺寸、信号线的条数以及排列等。例如，打印机有针式打印机、激光打印机、喷墨打印机等，不同打印机的打印方式各自不同，对打印头和纸张的移动的驱动方式也有所不同。

2）电气特性方面

数据在信号线上传递时，是以一定的电平值来表示二进制 0 和 1 的。电气特性定义每一条信号线的传递方向和有效电平范围。有的设备的电信号使用的是 TTL 电平标准，有的设备使用的是 CMOS 电平标准，这两者是不兼容的，其中 TTL 电路电源电压使用的是 5V，而 CMOS 电路电源电压使用的是 12V。另外，两者在表示逻辑 0 和逻辑 1 的电平值也是不相同的。

3）功能特性方面

功能特性定义了设备连接的每一条信号线的功能，如用于传递数据的信号线、用于传递地址的信号线、用于传递控制的信号线等。尤其对于种类繁多的外围设备来说，不同的设备所需的控制信号也各不相同，有的设备与主机之间采用中断传送方式，需要中断控制信号，有的设备与主机之间采用 DMA 传送方式，需要相应的 DMA 控制信号等。

4）数据格式方面

外围设备与主机之间的数据传送主要分为两种形式：串行传送和并行传送。串行传送是一位一位地进行的，而并行传送则是多位同时进行的，不同的设备并行传送的位数也会有所不同。

5）传输速度方面

不同外围设备在速度上的差异是非常明显的。有的设备的数据传输速率高达每秒几百兆字节（如磁盘），而有的设备则只有几十字节甚至更低。例如，键盘与 CPU 之间传输数据的速度取决于人的手指按键的速度，一个键对应一个字节的数据，1s 按 10 个键，数据传输速率也才 10B/s。

作为计算机硬件的核心——CPU，从功能上讲，它能够完成对所有其他硬件部件的控制。这种控制功能的实现是通过 CPU 芯片的引出脚与其他硬部件连接，并由 CPU 执行的指令产生的各种控制信号来完成的。但是，从上面讨论的有关设备之间的差异可以看出，一个 CPU 在设计时，无法做到与现在已有的和将来新开发出来的各种不同的外围设备直接相连。因此，计算机在设计时，针对与一些标准输入输出设备（这些设备往往是计算机的标配设备，如输入设备鼠标键盘、输出设备显示器、存储设备硬盘或光驱等）的连接，专门在主机板上设计了相应的接口电路，使得 CPU 可以通过这些接口电路实现对这些设备的控制。而一些计算机非标准配置的设备（如各种多媒体信息输入输出设备、过程控制设备等），或者通过计算机的一些标准接口（如串口、并口、USB 接口等）连接，或者使用专门的设备控制适配器与 CPU 连接，如图 5-1 所示。

无论是与计算机标准输入输出设备连接的标准接口还是与计算机非标准配置设备连接的专用接口统称为计算机的输入输出接口，简称 I/O 接口。也就是说，CPU 是通过 I/O 接口与各种外围设备连接的，并通过 I/O 接口实现对外围设备的控制功能。

1. I/O 接口的组成结构

不同设备的 I/O 接口在组成结构上有所不同，复杂程度往往会差异很大。但一般来说，从 I/O 接口的内部结构看，在大多数 I/O 接口中都包含一些数据寄存器、地址寄存器、状态寄存器、控制寄存器和相应的控制电路。而且，为了控制的灵活性和适应性，很多 I/O 接口中的寄存器往往是可编程的，即可以对 I/O 接口的功能、工作方式、操作方式、数据格式等进行预设置，以满足应用场合的要求。I/O 接口与 CPU 及设备连接的结构图如图 5-2 所示。

1）数据寄存器

数据寄存器保存 CPU 与设备之间交换的数据，又可以分成数据输入寄存器和数据输出寄存器。在接口电路连接输入设备时，需要从输入设备获取数据，数据从输入设备出来就暂时保存在数据输入寄存器，CPU 选择合适的方式进行读取。同样，当接口电路连接输出

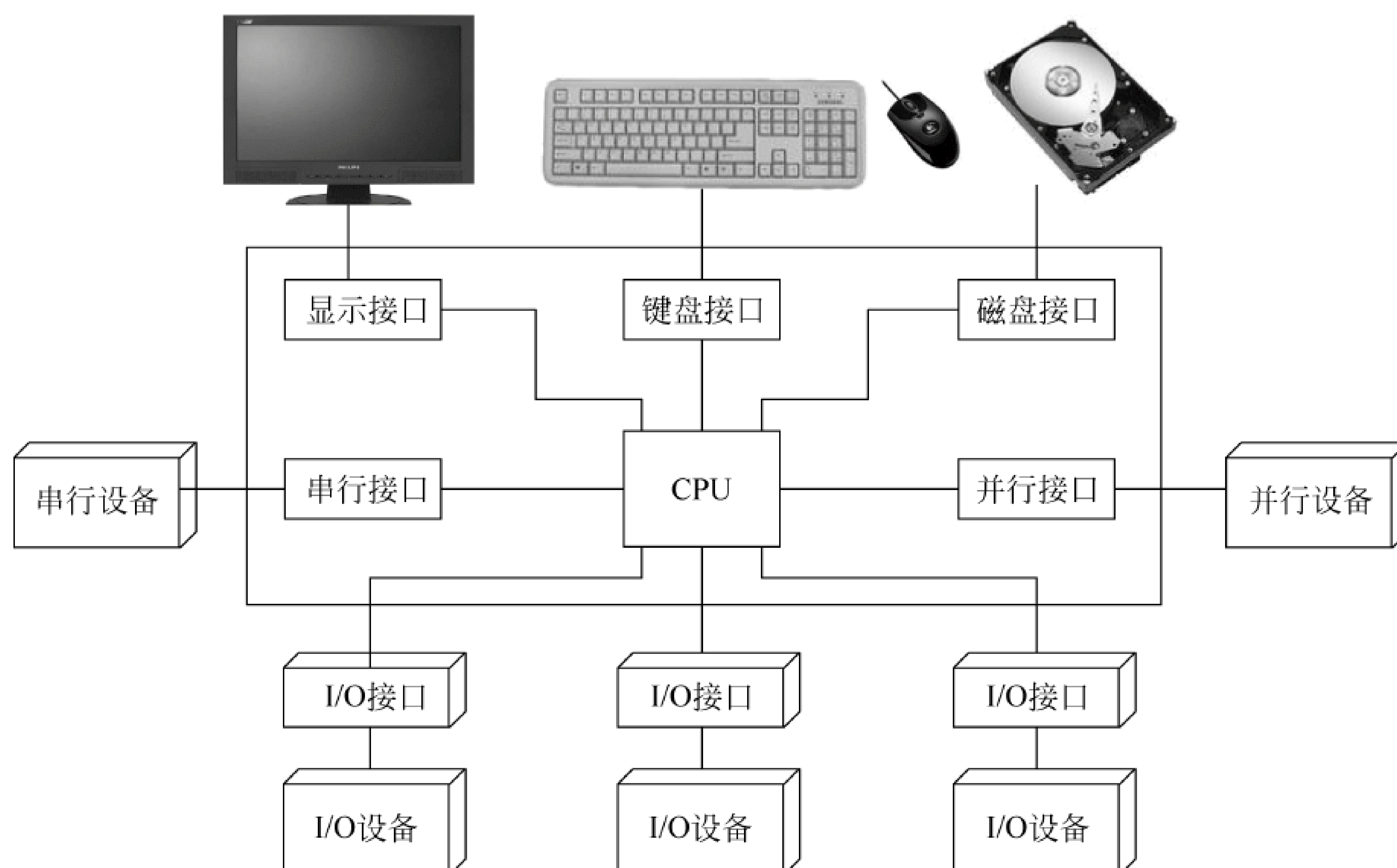


图 5-1　输入输出设备与接口

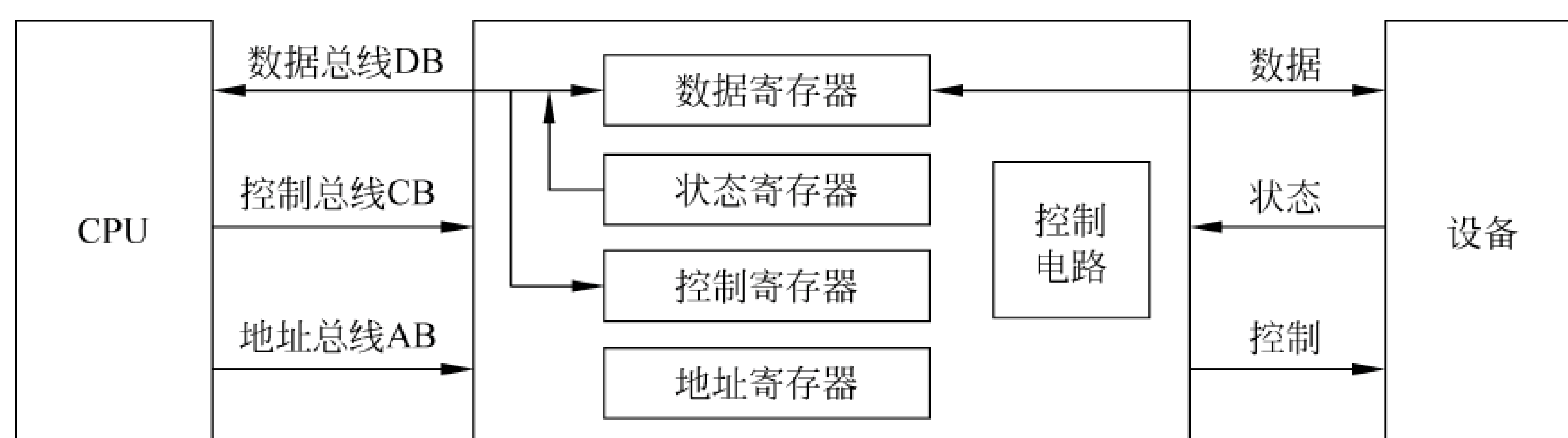


图 5-2　I/O 接口与 CPU 及设备连接结构图

设备时，CPU 发往输出设备的数据被临时保存在数据输出寄存器中，适时到达输出设备。很多设备既可以输入、又可以输出，可使用同一个寄存器或不同寄存器与 CPU 交换数据。

在 I/O 接口中设置数据寄存器，一方面可以用于暂存 CPU 与设备之间交换的数据；另一方面可以用于匹配 CPU 与设备之间的传输速度差异。例如，CPU 向一个慢速设备输出数据，CPU 每次首先将数据送 I/O 接口中的数据寄存器，然后等待 I/O 接口中的数据被传送给设备，再向 I/O 接口传送下一个数据。另外，I/O 接口中的数据寄存器还起到 CPU 与设备间数据格式的转换工作。例如，与 CPU 连接的是一个串行输出设备，则 CPU 每次将一个并行数据（字节或字）先送至 I/O 接口数据寄存器，由 I/O 接口控制将数据寄存器中的数据一位一位地移位送往设备。

2）状态寄存器

状态寄存器用于保存设备当前的工作状态信息。CPU 与设备交换数据，常常需要了解设备当前的工作状态，以便决定是否继续与设备进行数据的交换。例如，CPU 与慢速的设备进行数据交换时，若 CPU 只顾按自己的速度发送，而不管设备是否有能力接收，则会导致传输数据的丢失。通过在 I/O 接口中设置状态寄存器，及时地将设备的状态反映给

CPU,使 CPU 能够按照设备的接收能力或速度传输数据。

3）控制寄存器

控制寄存器用于保存 CPU 对 I/O 接口电路和设备进行控制或操作的相关信息。对于可编程的 I/O 接口来说,常常有多种工作方式或操作模式等可以选择,CPU 通过向控制寄存器写入相应的控制字实现对 I/O 接口及设备工作方式等的预设置。

4）地址寄存器

在有些 I/O 接口中还设置了地址寄存器,例如,在 CPU 与磁盘采用 DMA 数据传送方式的 DMA 接口电路中,就设置了地址寄存器,其初始值由 CPU 预置,指出与磁盘进行块数据交换的主存空间首地址,然后在 DMA 接口(又称 DMA 控制器)的控制下,完成块数据交换。

2. I/O 接口的功能

I/O 接口从简单到复杂,实现的功能各不相同。一般来讲,I/O 接口的功能包括以下几个方面。

1）数据的寄存和缓冲

I/O 接口中设置的数据寄存器,一方面起到数据暂存的作用;另一方面起到数据缓冲的作用。I/O 接口的数据缓冲用于匹配快速的 CPU 与相对慢速的设备之间的数据交换。例如,很多打印机是通过计算机的并行接口与主机相连的,在主机板的并行接口电路里设置了数据缓冲寄存器,同时打印机内部的控制电路也设计有一个数据缓冲区,CPU 传送来的打印数据首先被送到并行接口电路的数据寄存器中缓存,然后再传送到打印机的数据缓冲区,打印机的控制电路控制打印机按照一定的打印速度从数据缓冲区中取出数据打印。

2）对设备的控制和监测

一方面 I/O 接口能够接收来自 CPU 的控制命令,进行解释分析,生成对设备的控制信号;另一方面 I/O 接口能够对设备的工作状态进行实时监控,并随时将设备的状态提供给 CPU 查询等。

3）对设备的寻址

一个计算机系统中的外围设备很多,CPU 对设备的访问也是通过地址的方式进行的。在 I/O 接口电路里,除了一些寄存器和控制电路外,还往往有地址译码电路,CPU 通过 I/O 接口中的地址译码电路完成对设备的寻址和访问。

4）信号变换

外围设备大都是复杂的机电设备,其所需的控制信号和所能提供的状态信号往往同 CPU 的引脚信号不兼容,这包括信号使用的电平标准、信号的时序逻辑关系及数据格式等。因此,I/O 接口必须完成 CPU 与设备之间的这种信号转换功能,使 CPU 与设备能进行正常的连接和通信。

在微型计算机系统中,有些 I/O 接口是以标准芯片的方式提供用户使用的,它们往往具有可编程功能,以满足不同用户在不同应用场合的需要。

5.1.3 输入输出设备的编址与管理

在第 4 章讲到,CPU 对存储器的组织管理采用的是编址方式,即为每个存储单元分配

一个唯一的地址，CPU 通过给出单元地址访问相应的存储单元。在一个计算机系统中，所拥有的各种输入输出设备和存储设备很多，在这些设备以及设备的 I/O 接口中有很多可供 CPU 访问的寄存器，这些不同种类的寄存器称为 I/O 端口（Port）。CPU 对 I/O 端口的访问采用的是与访存类似的按地址访问方式，即为每一个 I/O 端口分配一个地址，又称为 I/O 地址或 I/O 端口号，CPU 通过给出 I/O 端口地址访问相应的 I/O 端口，也即访问相应的设备。

CPU 对 I/O 端口的编址方式主要有两种：一是独立编址方式；二是统一编址方式。

1. I/O 端口的独立编址方式

独立编址方式是指系统使用一个不同于主存地址空间之外的单独的地址空间，为外围设备及接口中的所有 I/O 端口分配 I/O 地址，如图 5-3(a)所示。

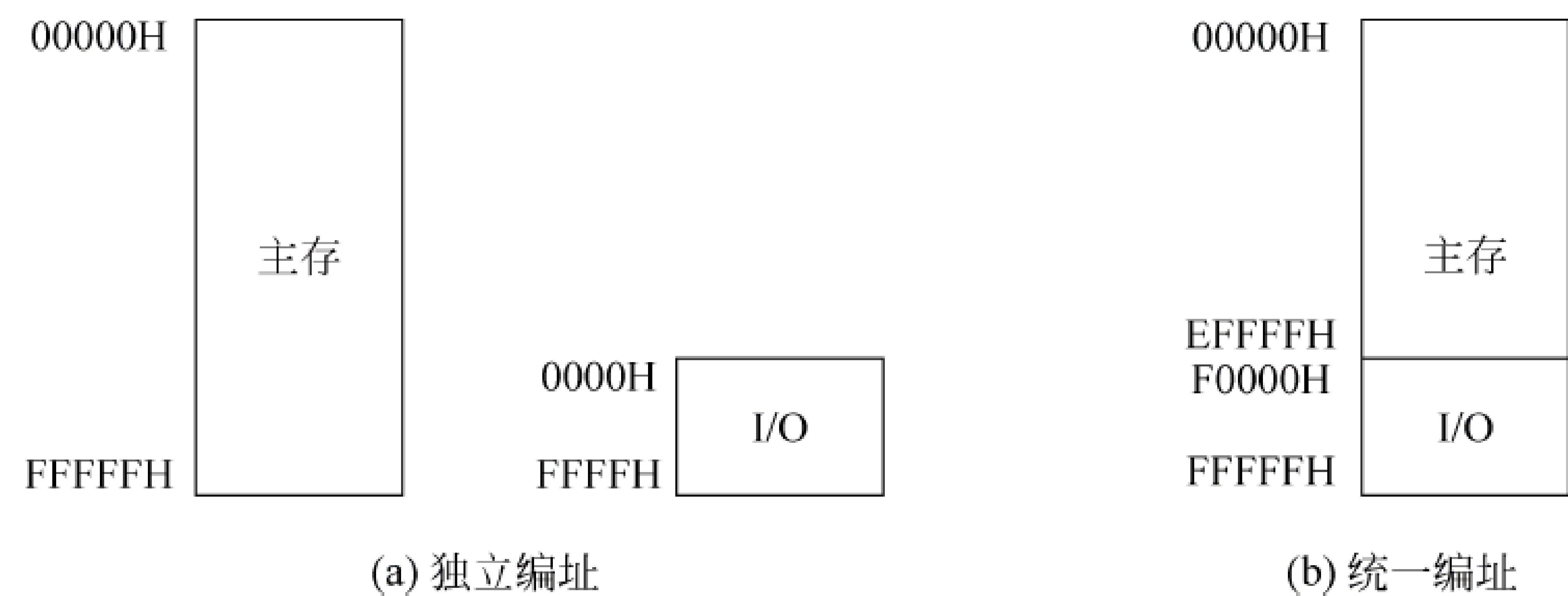


图 5-3 I/O 端口的编址方式

IBM PC 等系列计算机设置有专门的 I/O 指令，设备的编址可达 512 个，部分设备的地址码如表 5-1 所示。每种设备占用了若干个地址码，分别表示相应的设备控制器中的各个寄存器地址，各个寄存器的地址码不能重复，具有唯一性。

表 5-1 输入输出设备地址分配表

输入输出设备	占用地址数	地址码(十六进制)
硬盘控制器	16	320～32FH
软盘控制器	8	3F0～3F7H
单色显示器/并行打印机	16	3B0～3BFH
彩色图形显示器	16	3D0～3DFH
异步通信控制器	8	3F8～3FFH

在这种方式下，CPU 指令系统中有专门的用于与设备进行数据传输的输入输出指令，对设备的访问必须使用这些专用指令进行。例如，x86 系列的 CPU 有两条专用输入输出指令为

IN AL，PORT；输入指令——将 I/O 端口 Port 中的内容传送到 AL 寄存器中。

OUT PORT，AL；输出指令——将 AL 寄存器中的内容传送到 I/O 端口 Port 中。

执行 IN 指令时，CPU 的 I/O 读取 $\overline{\text{IOR}}$ 信号有效，产生 I/O 读取总线周期；执行 OUT 指令时，CPU 的 I/O 写 $\overline{\text{IOW}}$ 信号有效，产生 I/O 写总线周期。

独立编址方式的优点：一是I/O端口的地址没有占用主存的地址空间；二是I/O端口的地址码较短，地址译码器设计及实现简单，译码时间也较短。其缺点是：只能使用专用输入输出指令访问I/O设备，对I/O设备操作的程序设计灵活性较差。

2. I/O端口的统一编址方式

统一编址方式是指I/O端口与主存单元使用同一个地址空间进行统一编址，如图5-3(b)所示。

在这种方式下，CPU指令系统中无须设置专门的与设备进行数据传输的输入输出指令，I/O端口被当成主存单元同样对待，对主存单元进行访问和操作的指令可以同样用于对I/O端口的访问和操作。

统一编址方式的优点：可以使用访存指令访问I/O，对I/O设备操作的程序设计灵活性较好。其缺点是：I/O端口的地址占用了主存的部分地址空间，对I/O端口访问的地址译码更加复杂。

5.2 输入输出控制方式

从前面的讨论大家已经了解到，不同的设备在很多方面都存在差异，因此，作为机器的输入输出系统来说，必须针对不同的设备采取不同的输入输出控制方式，以最大程度发挥主机CPU和设备的效率。

5.2.1 程序控制方式

程序控制方式是指主机与设备间的数据传输是通过CPU执行一段软件程序，在程序的控制下完成输入输出操作。程序控制方式根据设备的不同又分为无条件传送控制方式和查询式传送控制方式。

1. 无条件传送控制方式

有些简单设备，如发光二极管(Light-Emitting Diode，LED)和数码管、按键及开关等，它们的工作方式十分简单，相对CPU而言，其状态很少发生变化。例如，数码管，只要CPU将数据传给它，就可立即获得显示；又如电子开关，其状态或者为闭合或者为打开，CPU可以随时对其状态进行读取。因此，当这些设备与CPU交换数据时，可以认为它们总是处于就绪(Ready)状态，随时可以进行数据传送，这就是无条件传送，有时也称它为立即传送或同步传送，其程序控制流程如图5-4所示。

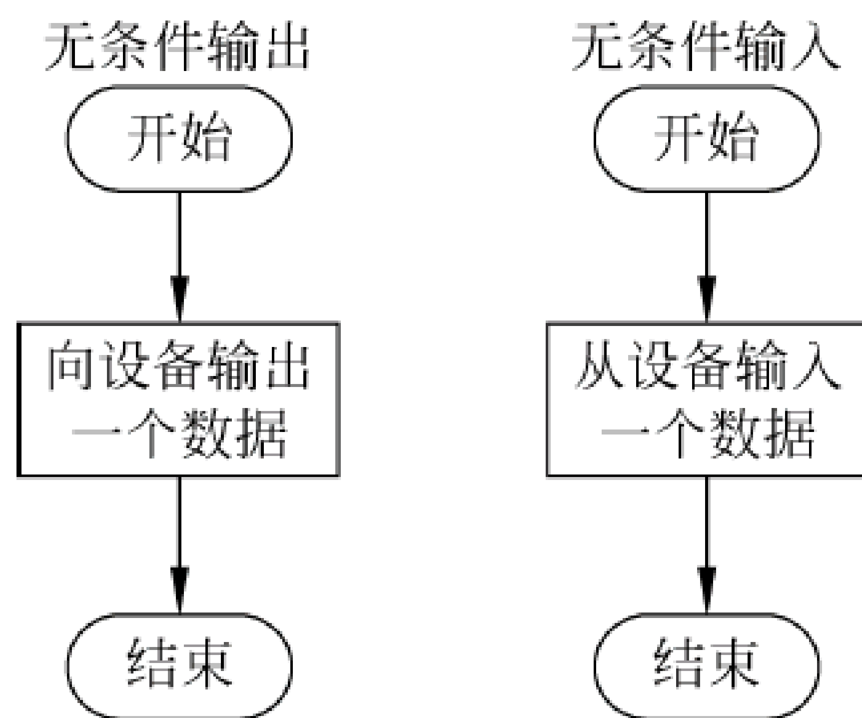


图5-4 无条件传送程序控制流程

用于无条件传送的I/O接口电路十分简单，接口中只考虑数据缓冲，不考虑信号联络。实现数据缓冲的器件可以是三态缓冲器或锁存器。

图5-5给出了一个无条件输入输出的接口例子，其中无条件输入接口电路连接一个开关电路，无条件输出接口

电路连接发光二极管(LED)。

三态缓冲器 74LS244 构成数据输入端口,它连接 8 个开关 $K_0 \sim K_7$,开关的输入端通过电阻连到高电平上,另一端接地。这样,当开关打开时缓冲器输入端为高电平(逻辑 1);开关闭合时缓冲器输入低电平(逻辑 0)。

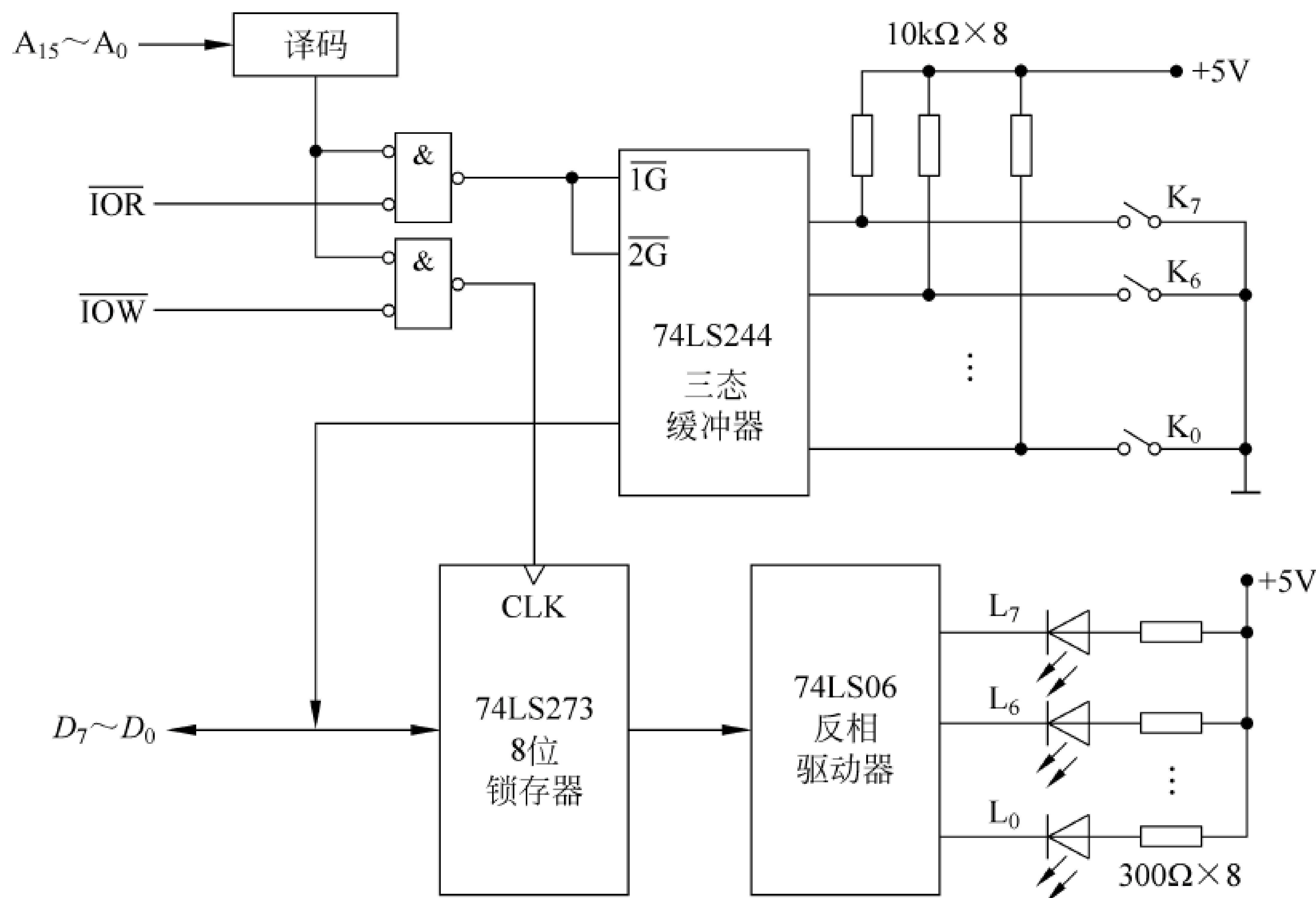


图 5-5 无条件传送

当 CPU 执行输入指令时,产生读控制$\overline{IOR}$信号低有效。译码输出和读控制同时低有效,使得三态缓冲器控制端低有效;开关的当前状态被三态缓冲器传输到数据总线 $D_7 \sim D_0$,进而传送给 CPU,其中某位 $D_i = 0$,说明开关 K_i 闭合;$D_i = 1$,说明开关 K_i 断开。

8 位锁存器 74LS273(无三态控制)构成数据输出端口。当其时钟控制端 CLK 出现上升沿时锁存数据,被锁存的数据输出,经反相驱动器 74LS06 驱动 8 个发光二极管($L_7 \sim L_0$)发光。74LS06 是集电极开路(OC)输出,它的每个输出线需要通过电阻挂到高电平上。当 CPU 的某个数据总线 D_i 输出高电平(逻辑 1)时,经反相为低电平接到发光二极管 L_i 负极,发光二极管正极接高电平。这样,二极管形成导通电流,发光二极管 L_i 将点亮。当 CPU 输出 D_i 低电平,对应发光二极管 L_i 不会导通,将不发光。

当 CPU 执行输出指令时,产生写控制$\overline{IOW}$信号低有效。译码输出和写控制同时低有效,使得 8 位锁存器控制输入 CLK 为低;经过一个时钟周期,译码输出或写控制无效将使得 CLK 恢复为高。在 CLK 的上升沿,8 位锁存器将锁存此时出现在其输入端,由 CPU 传送到数据总线 $D_7 \sim D_0$ 上的输出数据,控制发光二极管的发光和不发光。

2. 查询式传送控制方式

查询式传送控制方式也称为异步传送,它是指当 CPU 需要与外围设备交换数据时,首先查询设备的状态,只有在设备准备就绪时才进行数据传输。与无条件传送方式相比,查询式传送增加了一个传送前查询设备状态的环节。查询式输入和输出程序控制流程如图 5-6 所示。

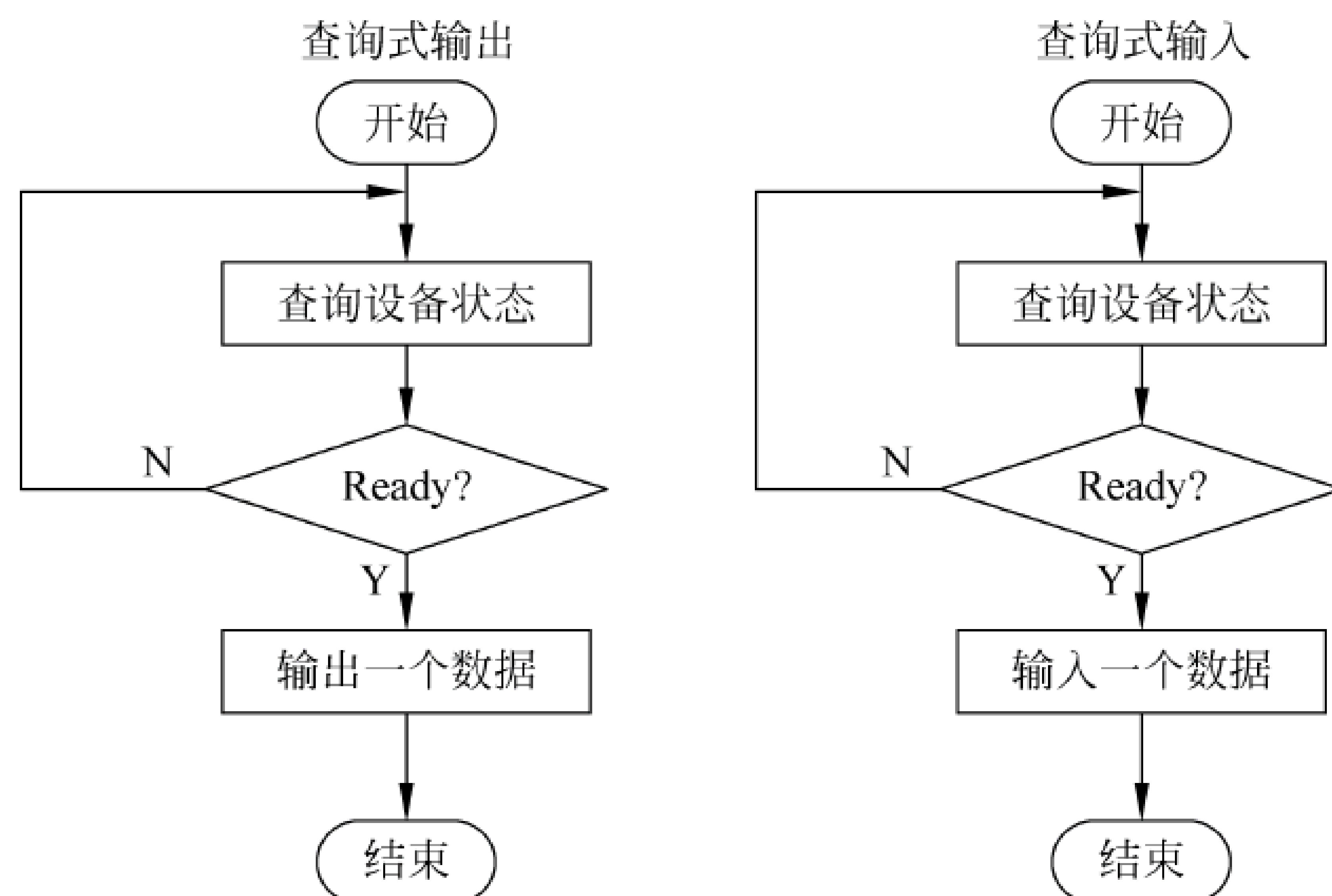


图 5-6　查询式传送程序控制流程

1）查询式输入接口

如图 5-7 所示是一个采用查询方式输入数据的 I/O 接口电路示意图。图中 8 位锁存器与 8 位三态缓冲器构成数据输入寄存器，即数据端口。它一侧连接输入设备，另一侧连接系统的数据总线。1 位锁存器和 1 位三态缓冲器构成状态寄存器，即状态端口，其输出端连接到数据总线的最低位 D_0。

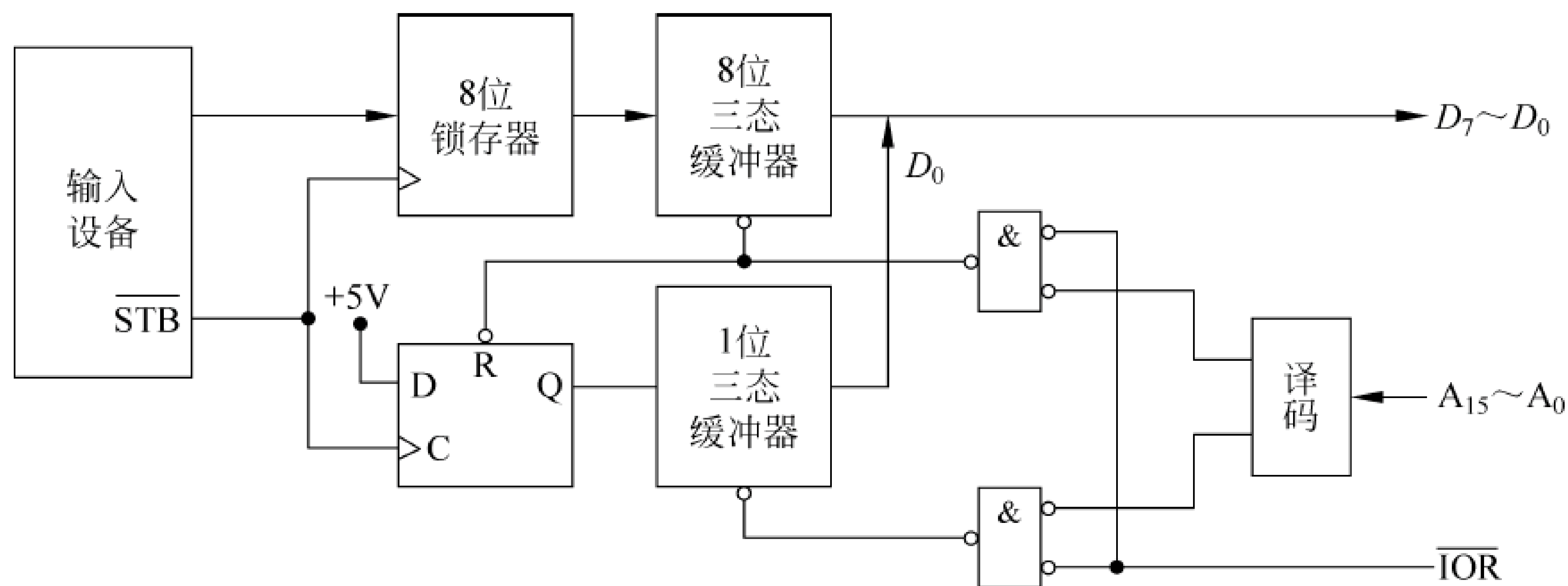


图 5-7　查询式输入接口

在输入设备通过选通信号$\overline{STB}$将数据送入数据寄存器的同时，将 D 触发器置 1（因为其输入端 D 总是为高电平），说明数据寄存器中已经有设备送来的数据，可以提供给 CPU 读取，也就是表示设备就绪。

CPU 可以随时读取状态端口来查询设备的状态。如果 $D_0=1$，说明输入数据已就绪，此时，CPU 读取数据端口便得到设备提供的数据。读取数据产生的控制信号，还被连接到 D 触发器的 R 复位端（低电平有效），该复位信号将触发器输出 Q 置为 0，表示数据已被取走。如果检测到 $D_0=0$，说明输入数据尚未就绪，程序应继续查询。

2）查询式输出接口

如图 5-8 所示是一个采用查询方式输出数据的接口电路示意图。8 位锁存器构成数据输出寄存器，即数据端口，它一侧连接系统的数据总线，另一侧连接输出设备。1 位锁存器

和1位三态缓冲器构成状态寄存器，即状态端口，其输出端连接到数据总线的最高位 D_7。

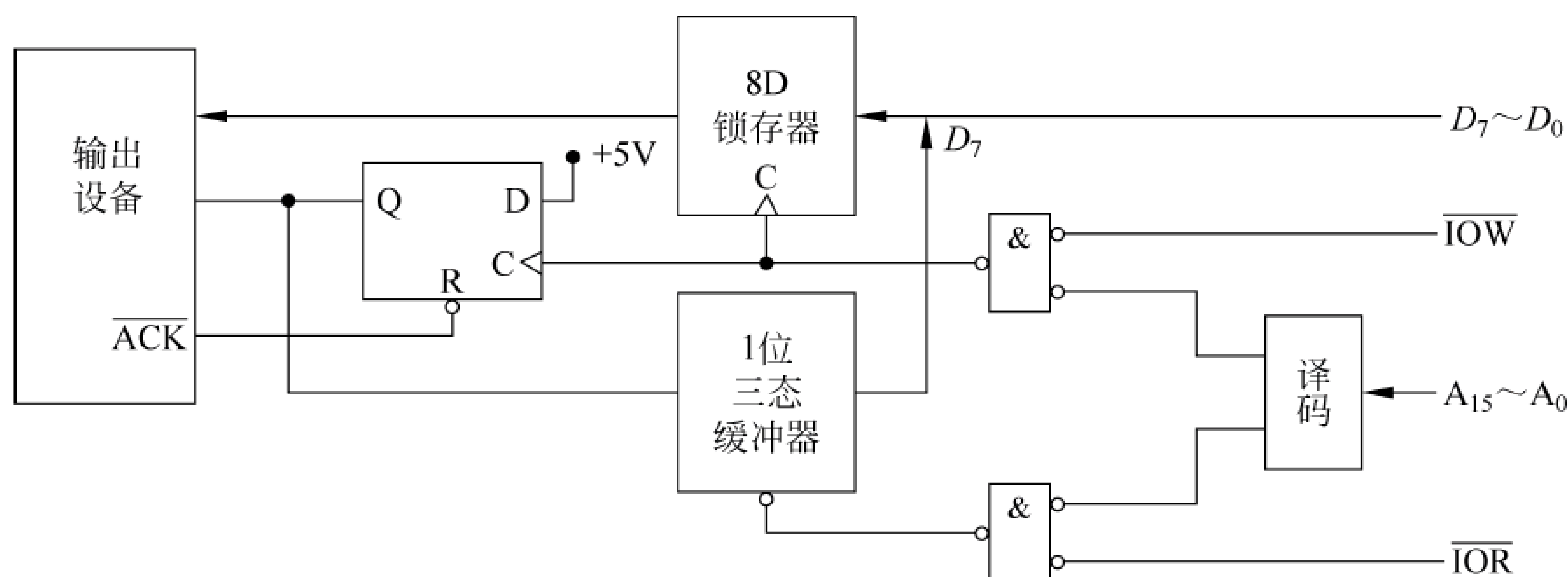


图 5-8　查询式输出接口

当CPU要输出数据时，首先查询状态端口。若 $D_7=0$，表示设备准备就绪可以接收数据。此时，CPU可将数据写入数据端口，写入数据产生的控制信号也作为 D 触发器的控制信号，它将 D 触发器置1，以便通知外围设备接收数据。若 $D_7=1$，说明接口电路中的上一个数据尚没有被设备取走，CPU不能送下一个数据，而只能继续查询设备状态，直到设备准备就绪。

输出设备可根据状态锁存器的输出信号 Q 接收数据。若为1，表示CPU已送来一个数据，于是，设备可以从数据寄存器中将数据取出，并待数据处理结束时，给出应答信号 $\overline{ACK}$，该信号将状态寄存器重新复位为0，表示设备准备就绪，可以接收下一个数据了。

5.2.2　中断控制方式

上述的程序控制方式对于在单用户单道程序系统环境下对简单设备的输入输出控制是有效的。在这种环境下，用户程序可以占用全部系统资源，包括CPU、存储器和输入输出设备。当用户程序需要进行输入输出操作时，CPU执行用户的输入输出程序，控制与设备的数据交换。但是，在多用户多道程序系统中，若仍采用这种控制方式，某一道程序需要占用CPU进行输入输出操作，而其他程序只能等待，这对CPU这一宝贵资源将是极大的浪费。尤其是当CPU控制与慢速的设备进行数据交换时，一方面CPU大部分时间处于等待设备准备就绪的空置状态，另一方面其他程序因得不到CPU而不能运行，使得CPU的有效利用率很低。为解决这种矛盾，计算机设计者提出了中断控制方式。

1. 中断的基本概念

老师正在教室上课，一位迟到的学生推门进来，老师停下正在讲的课，目送着学生在座位上就坐，然后继续上课，这一过程就是中断。

对于计算机系统来说，**中断**是指CPU正在运行一个程序时发生了某种非预期的事件，CPU暂停正在运行的程序，转而执行对这一事件进行处理的程序（称为中断服务程序），完成后再返回原程序继续运行的过程，如图5-9所示。

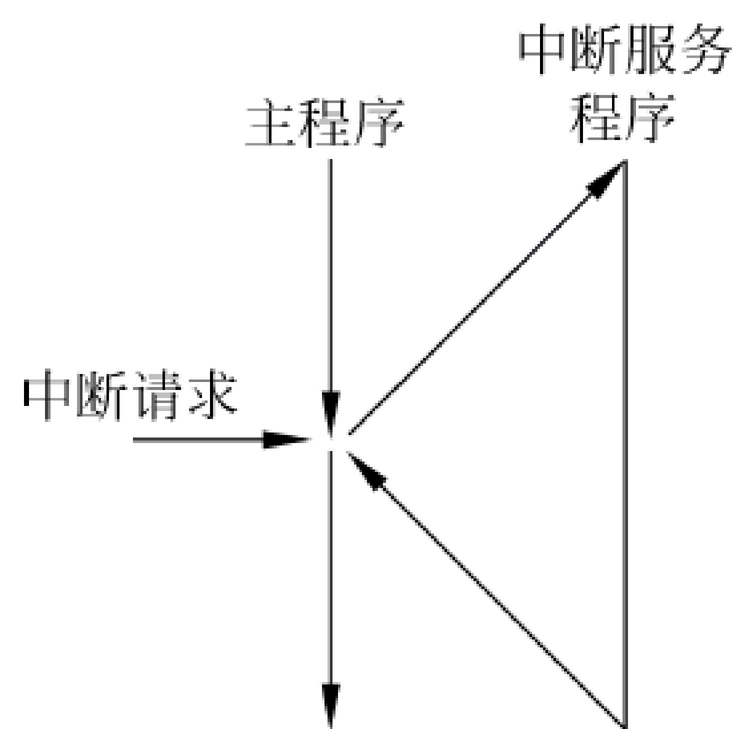


图 5-9　中断的概念

引起中断的事件称为**中断源**。对于计算机系统来说，中断源分为外部中断和内部中断。**外部中断**主要是指由计算机外围设备、系统定时时钟及人工干预等外部事件引起的中断，通过设备产生的外部中断，能使 CPU 与设备间进行中断方式的数据传输，这也是本小节主要讲述的内容。内部中断主要包括指令中断和故障中断。指令中断是由软件指令引起的，设置指令中断的目的通常是为用户程序提供对系统资源的访问，例如，x86 CPU 指令系统提供了一条软中断指令 INT，执行该指令后，系统会转入执行一段驻留在主存中的系统程序，该程序主要完成对系统某一资源的访问服务。故障中断主要是指由系统软硬件故障引起的中断，如内存校验故障、电源掉电、除零错误、算术溢出、内存越界、指令非法、虚拟存储器页面失效等。计算机中断源的类型归纳总结如图 5-10 所示。

2. 中断控制的基本原理

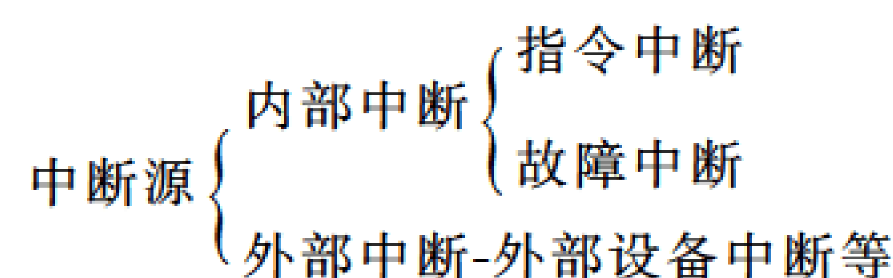


图 5-10 计算机中断源的类型

前面已经提到，程序控制方式在多道程序系统中会因为某一程序长时间占用 CPU 进行输入输出操作而浪费 CPU 资源，而中断控制方式则可以有效提高 CPU 利用率。下面通过对比程序控制方式和中断控制方式下 CPU 控制打印机打印输出的过程，阐述中断控制的基本原理。

图 5-11(a)和图 5-11(b)分别给出了程序控制方式和中断控制方式这两种方式下打印机的打印输出过程。

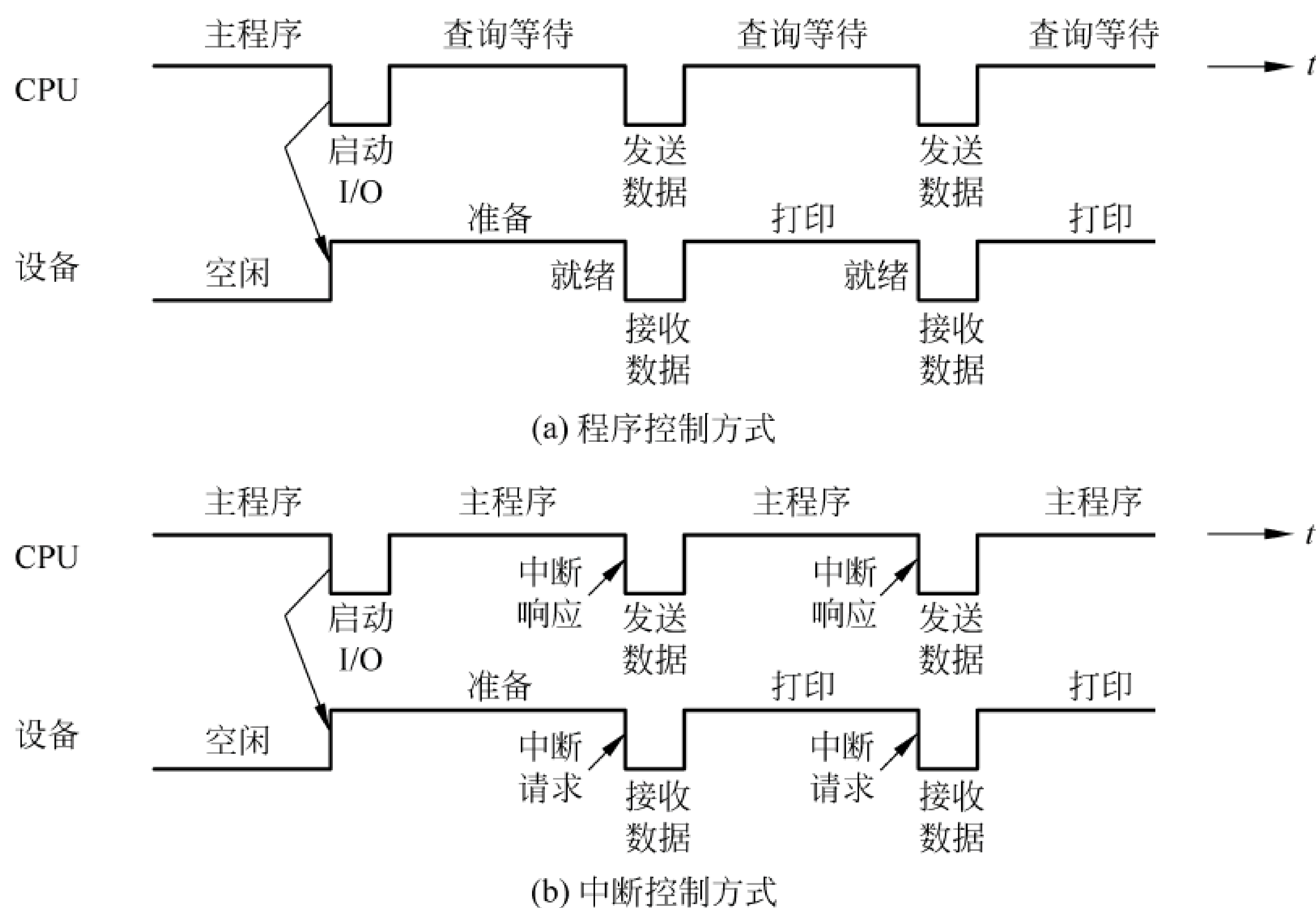


图 5-11 程序控制方式和中断控制方式下打印输出的过程对比

从图 5-11 可以看出，在程序控制方式下，当 CPU 执行的一个主程序要打印输出数据时，首先启动打印机，使打印机进入打印准备工作状态。在打印机准备的过程中，CPU 一直处于查询等待状态，即查询打印机是否准备就绪。当 CPU 查询到打印机已做好打印准备时，便向打印机传送第一个数据，打印机接收到该数据后进行打印。在打印机打印的过程中，CPU 又开始进入查询等待状态，等到打印机打印完一个数据后，可以接收下一个数据

时,CPU 再向打印机发送下一个数据。如此重复,直到将所有数据打印完。一般来讲,CPU 向打印机传送一个数据的时间远比打印机打印一个数据所花的时间少得多,而在上述过程中,当打印机进行打印操作时,CPU 一直在执行一段查询打印机状态的循环控制程序,无法执行其他程序,这对 CPU 资源是极大的浪费。

而在中断控制方式下,情况就不一样了。当 CPU 执行的一个主程序要打印输出数据时,首先启动打印机,使打印机进入打印准备工作状态。在打印机准备的过程中,CPU 可以由系统调度去执行其他主程序。当打印机做好打印准备时,向 CPU 发出一个中断请求信号,CPU 接收到该请求后,暂停正在执行的主程序,向打印机传送一个数据,然后返回被中断的主程序继续执行。打印机接收到数据后进行打印,打印完成后,又向 CPU 发出中断请求,CPU 响应中断,向打印机传送下一个数据进行打印。如此重复,直到将所有数据打印完。从这一过程可以注意到,在打印机打印的同时,CPU 可以被调度执行其他主程序,而无须查询等待,使 CPU 的利用率得到了提高。

通过对以上两种不同控制方式控制打印机打印输出的过程进行比较可以看出它们的不同之处:

(1) 程序控制方式下,CPU 是通过查询方式了解打印机的状态的;而中断控制方式下,CPU 是通过中断方式了解打印机的状态的。

(2) 程序控制方式下,CPU 和打印机之间是串行工作的;而中断控制方式下,CPU 和打印机可以并行工作。

(3) 程序控制方式对于单用户单道程序系统来说是有效的;而中断控制方式对于多用户多道程序系统来说可以大大提高 CPU 的利用率。

3. 中断处理过程

一个中断的处理过程是由中断源的中断请求引起的。在一个实现了中断系统功能的机器中,CPU 在执行一个主程序时,每执行完一条指令都会检查是否有中断请求发生,若没有,则继续执行原程序,若有,则在条件满足的情况下,暂停正在执行的程序,对中断请求进行响应。在中断响应过程中,中断系统要识别是哪一个中断源发出的中断请求,在有多个中断源同时发出中断请求的时候还要决定首先响应哪一个中断源的请求。在确定了要响应的中断源后,进入对该中断源的中断请求进行处理的中断服务程序去执行,执行完成后返回原来被中断的程序继续执行。中断系统的中断处理过程如图 5-12 所示。

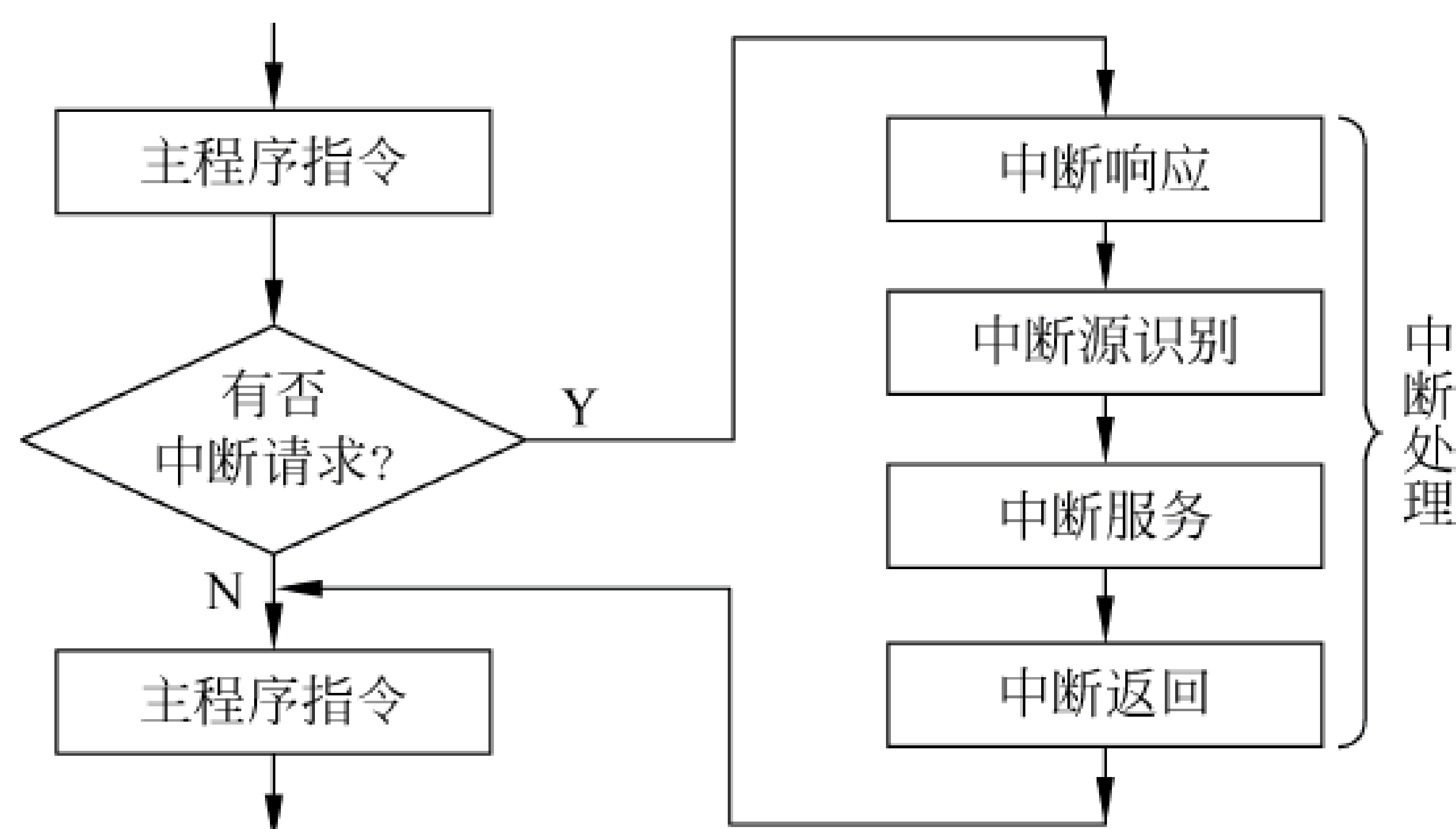


图 5-12　中断处理过程

1）中断请求的建立

CPU 在设计时，其对外引脚通常会有一条或多条来自中断源的中断请求信号线，CPU 通过对这些引脚信号的状态来判别是否有外部中断源中断请求的发生。例如，8086/8088 CPU 有两条引脚 INTR 和 NMI，其中 INTR（INTerrupt Request）主要用于定时时钟和外围设备中断源的中断请求，NMI（Non-Masked Interrupt）用于一些系统硬件故障中断源的中断请求等。

在中断系统中，外部中断源及硬件故障中断源的中断请求是由硬件实现的，当某中断源要向 CPU 发出中断请求时，首先通过硬件方式为其建立和保持一个中断请求信号。通常是在其中断接口电路里设置一个“中断请求触发器”，当中断源有中断请求时（对设备来说就是设备准备就绪，对故障中断源来说就是发生了硬件故障），将接口中的中断请求触发器置位，中断请求触发器的输出将作为发往 CPU 的中断请求信号，如图 5-13（a）所示。

为了控制的灵活性，通常在中断接口电路中还会设置一个中断屏蔽触发器，其作用是对中断源的中断请求进行屏蔽和开放。中断系统允许在程序中对中断屏蔽触发器进行设置，以决定在程序的执行过程中哪些中断源允许请求中断，哪些中断源不允许请求中断，如图 5-13（b）所示。

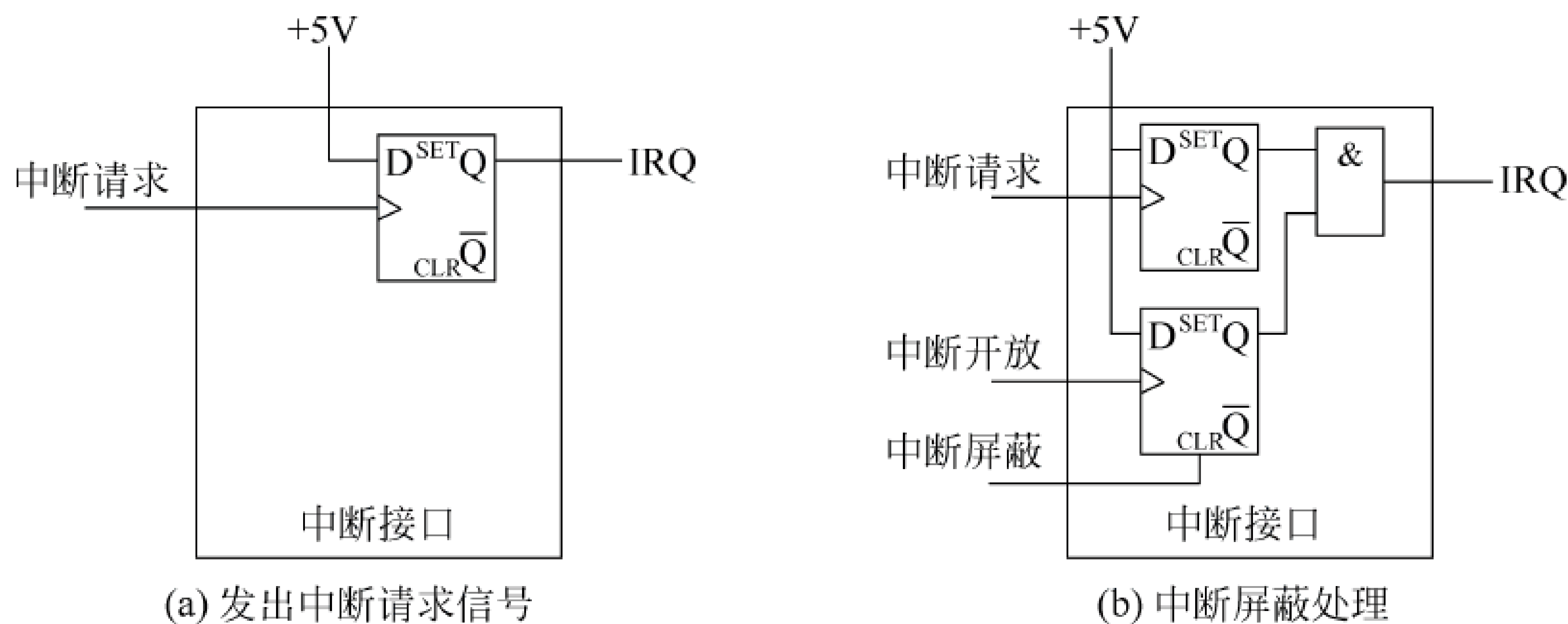


图 5-13 中断请求与中断屏蔽

在一个机器系统中，往往有多个中断源，将这些中断源的中断请求触发器合在一起便构成了一个中断请求寄存器，而这些中断源的中断屏蔽触发器合在一起构成了一个中断屏蔽寄存器。其中，中断屏蔽寄存器作为 I/O 端口可供 CPU 访问，在程序中通过将一个中断屏蔽字写入该端口，便可实现对中断源的屏蔽和开放功能。

2）中断响应

CPU 在其中断请求线上检测到外部中断源的中断请求时，并不是都会给予响应的。通常，大多数 CPU 内部都设置了一个中断允许触发器，该触发器的作用是对中断请求线上来的中断请求进行禁止和允许。中断允许触发器可以通过 CPU 提供的指令进行设置，例如，8086/8088 CPU 提供了两条指令 STI 和 CLI，前一条指令称为开中断指令，用于将 CPU 内部的中断允许触发器置 1，即中断允许或开中断；后一条指令称为关中断指令，用于将中断允许触发器清 0，即中断禁止或关中断。在 CPU 处于中断允许状态时，可以对来自中断请求线上的中断请求进行响应；而 CPU 处于中断禁止状态时，则对来自中断请求线上的中断

请求不予响应。

当然，并不是所有中断源的中断请求都能被 CPU 禁止的，尤其是一些比较紧迫的事件的中断，如因电源掉电、内存校验出错等导致的硬件故障中断，CPU 必须立即响应。因此，多数 CPU 在引脚设计上会设置多条中断请求信号线，一些用于可屏蔽的中断请求，一些用于非屏蔽的中断请求（如 8086/8088 CPU 的 INTR 和 NMI）。

一旦 CPU 响应了中断，便进入中断响应周期。在中断响应周期里，中断系统主要完成以下三项功能：

（1）关中断和保护断点。CPU 响应中断时，会自动执行一条中断隐指令，一方面将中断允许触发器清零，即关中断；另一方面将 CPU 内部的指令指针和程序状态字（PSW）等压入堆栈。当前指令指针指向的是下一条要执行的指令，又称为断点，将指令指针压入堆栈的目的是为了使在中断服务程序执行完成后能正确返回到当前被中断的程序的下一条指令继续执行。而程序状态字（PSW）记录的是当前指令执行完成后程序和机器的状态，将 PSW 压入堆栈的目的是为了保证中断返回后 PSW 能恢复成被中断前的状态。

（2）进行中断源的识别。一个机器的中断源和 CPU 的中断请求线往往不是一一对应的，换句话说，CPU 的一条中断请求输入线会对应多个中断源的中断请求输出。当 CPU 检测到一条中断请求信号有效时，它可以判定外部中断源有了中断请求，但却无法确定具体是哪一个中断源发出的请求。因此，在中断响应周期，CPU 要对发出中断请求的中断源进行识别。另外，在某一时刻，有可能同时有两个或两个以上的中断源向 CPU 发出中断请求信号，而 CPU 一次只能响应一个中断源的请求。在这种情况下，CPU 除了进行中断源的识别外，还要根据一定的规则选择其中一个进行响应。

（3）形成中断源中断服务程序的入口地址。

每一个中断源都对应有一段驻留在内存中的软件程序，称为中断服务程序，该程序的功能是完成中断源需实现的功能。例如，打印机是一个中断源，它所对应的中断服务程序的功能是实现 CPU 向打印机传输数据。在中断响应周期，识别中断源后，最后还需形成该中断源对应的中断服务程序在内存的入口地址，以便 CPU 从当前被中断的主程序转入中断服务程序执行。

3）中断源识别

中断源识别的任务是确定某次中断响应具体该响应的是哪个中断源。中断源识别的方法很多，常用的方法主要有软件查询法、硬件查询法和中断向量法等。

软件查询法是通过执行一段软件查询程序，对中断请求寄存器的状态逐位判断，从而确定某次该响应的是哪个中断源。前面讲到，将各中断源接口电路中的中断请求触发器合在一起构成一个中断请求寄存器，也就是说，中断请求寄存器的每一位就对应了一个中断源的中断请求状态。将中断请求寄存器的内容读出，按某一种顺序一位一位进行判别，遇到第一个“1”，这一位所对应的中断源就是本次 CPU 识别响应的中断源。

软件查询法的优点：一是实现容易，无须额外的硬件；二是控制灵活，查询顺序很容易调整变化，这一顺序也决定了优先响应的中断源的顺序。其缺点是：速度慢，只适合一些对中断处理速度要求不是很高的场合。

硬件查询法是通过专门的硬件电路实现中断源识别。一种实现中断源识别的串行排队链路如图 5-14 所示。

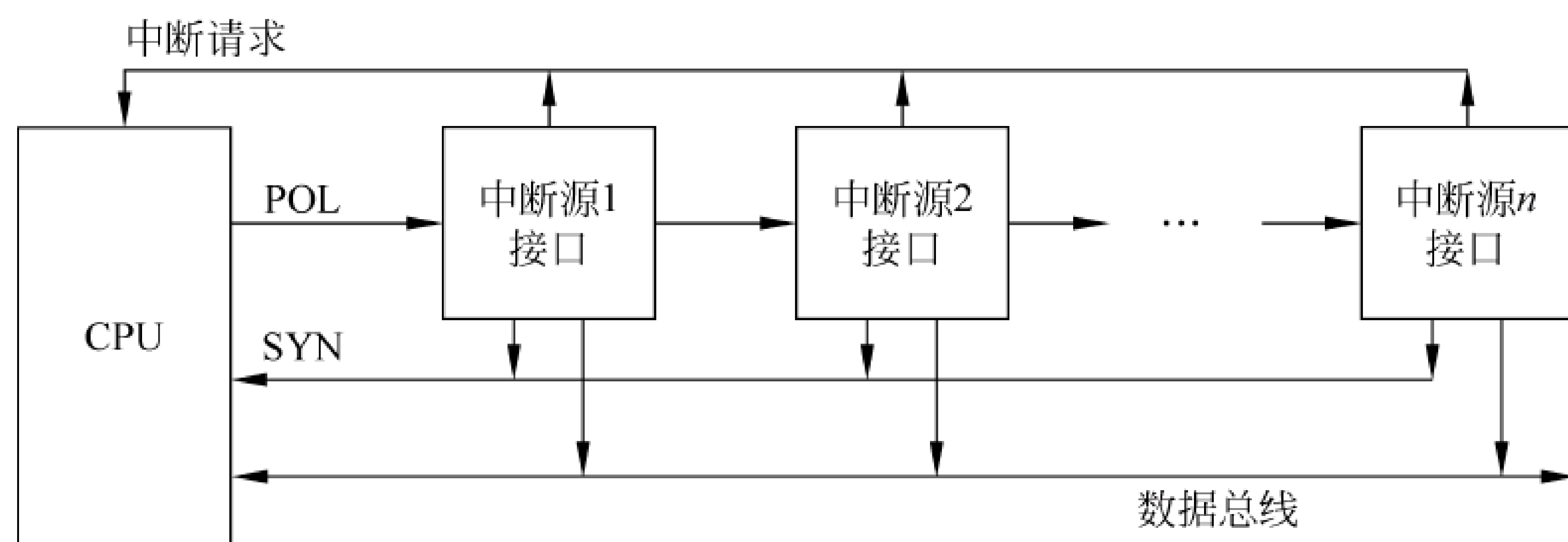


图 5-14 串行排队链中断源识别

在中断响应周期，CPU 发出查询信号 POL，沿着串行排队链依次经过各中断源接口。当 POL 到达某一中断源接口时，如果该中断源没有中断请求，则将 POL 信号继续往下传；如果该中断源有中断请求，则 POL 信号不再往下传，接口向 CPU 发出回答信号 SYN，同时形成中断源的中断服务程序入口地址，经数据总线传送给 CPU。

中断向量法是一种通过硬件控制电路形成一个所识别的中断源的中断向量号，并由此中断向量号实现中断响应的方法。在这种方法中，每个中断源对应有一个中断向量号，中断向量号对应一个中断向量，即中断服务程序入口地址，将所有中断向量集中存放在内存中的一片固定区域中。在中断响应周期，首先由一个专门的中断控制电路进行中断识别，并形成一个对应该中断源的中断向量号；然后将此中断向量号传送给 CPU；最后由 CPU 依据中断向量号生成该中断源的中断向量在内存中的首地址，从这一地址单元中即可取出中断服务程序的入口地址，如图 5-15 所示。

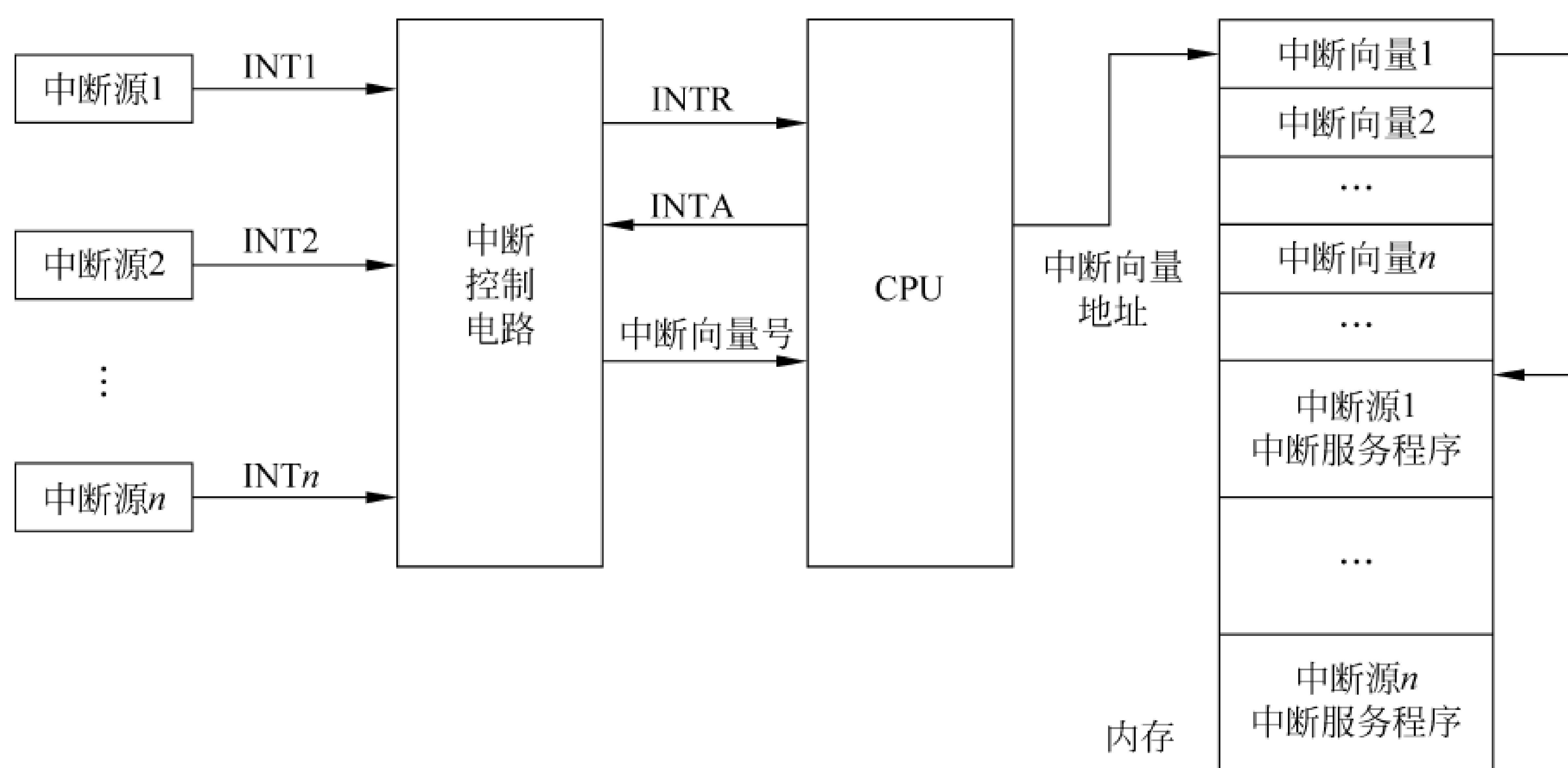


图 5-15 向量中断示意图

x86 CPU 采用的就是中断向量法，其中断响应过程如下：

① 当某一中断源 i 需要申请中断时，向中断控制电路发出一个请求中断信号 INTi。

② 在该中断源的中断请求未被屏蔽的情况下，中断控制电路向 CPU 发出中断请求信号 INTR。

③ 在 CPU 处于开中断的情况下，CPU 响应中断，向中断控制电路发回一个中断响应

信号 INTA。

④ 中断控制电路完成中断源的识别，并将中断源的中断向量号通过数据总线传送给 CPU。

⑤ CPU 依据此中断向量号计算得到中断向量地址，并从此地址单元中取出该中断源对应的中断服务程序入口地址。

4）中断服务

CPU 在中断响应周期获取到中断服务程序入口地址后，便可转入中断服务程序执行。一般来讲，中断服务程序包括以下几个过程（见图 5-16）。

① 保护现场。所谓现场是指主程序执行完当前指令时的一些寄存器内容等。由于中断服务程序在执行过程中有可能用到主程序使用到的一些寄存器，因此必须在执行中断服务程序前将这些寄存器的内容保护起来。保护现场的具体方法就是将 CPU 所有程序可用的寄存器内容压入堆栈，以便从中断服务程序返回主程序时再将这些寄存器的内容恢复到中断前的状态。

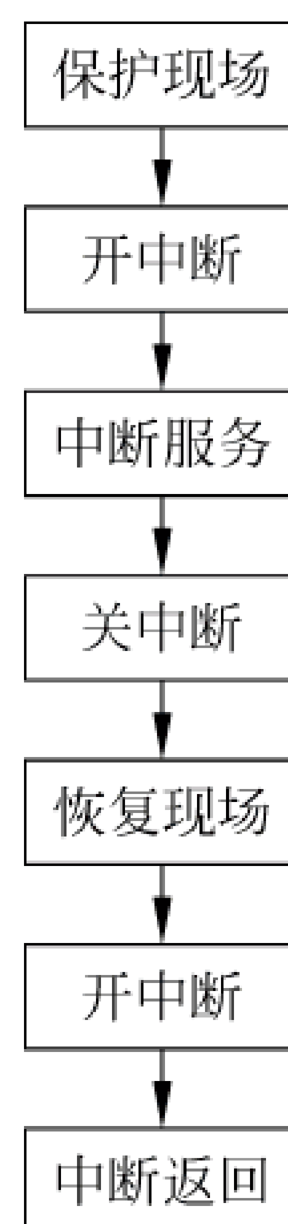


图 5-16　中断服务程序

② 开中断。前面讲到，CPU 响应中断时执行了一次硬件自动关中断的操作，这次关中断的目的是阻止在保护现场的过程中再次被中断。如果本次执行的中断服务程序在后续的中断服务过程中允许其他的中断，在这里就必须开中断。开中断的方法是执行一条开中断指令，将 CPU 的中断允许触发器置位。

③ 中断服务。其实，中断处理的核心就是执行这里的中断服务，在此之前和之后的所有操作都是为这里的中断服务提供支持的。中断服务的内容就是完成中断源的功能。例如，键盘作为计算机的标准输入设备，它与 CPU 之间采用的是中断控制方式进行数据传输。每当用户在键盘上按一个键，就会由键盘接口电路产生一个中断请求信号发往 CPU，CPU 响应键盘中断后进入中断服务程序。在键盘中断服务程序里的中断服务主要实现对用户所按键的识别，并根据所按键实现相应的功能。

④ 关中断。这次的关中断是在程序中使用关中断指令将 CPU 中的中断允许触发器清零，禁止一切可屏蔽中断，使后续的恢复现场工作不再被新的中断源中断。

⑤ 恢复现场。将中断服务程序开始时保护起来的寄存器内容恢复到中断响应前的状态，若保护现场使用的是入栈的方法，则恢复现场应使用相应的出栈操作。

⑥ 开中断。再次使用开中断指令将 CPU 的中断允许触发器置位，以便该中断服务程序执行完成后系统恢复到正常中断工作状态。

⑦ 中断返回。CPU 的指令系统通常会提供一条中断返回指令，中断服务程序结束前执行中断返回指令，执行该指令的结果是将系统中断响应保护断点时压入栈的内容出栈，一是恢复 PSW 值，二是恢复主程序被中断时的指令指针值，从而将程序控制返回到主程序被中断的指令继续执行。

4. 单级中断和多级中断

一个机器系统中有多个中断源，但 CPU 一次只能响应和处理一个中断源的中断请求。

当某一时间有两个或两个以上的中断源同时发出中断请求时，中断系统就必须从中选择一个进行响应，选择的依据就是各个中断源的**中断优先权**。中断系统可以对各中断源依据其发生的中断事件的重要性和紧迫性预先设定中断优先权，也可以由用户根据程序运行的需要灵活设置。

如果机器系统的中断源很多，还可以在中断优先权的基础上进一步分级，高一级的任何一个中断源的优先权都比低一级的任何一个中断源的优先权高，如图 5-17 所示。

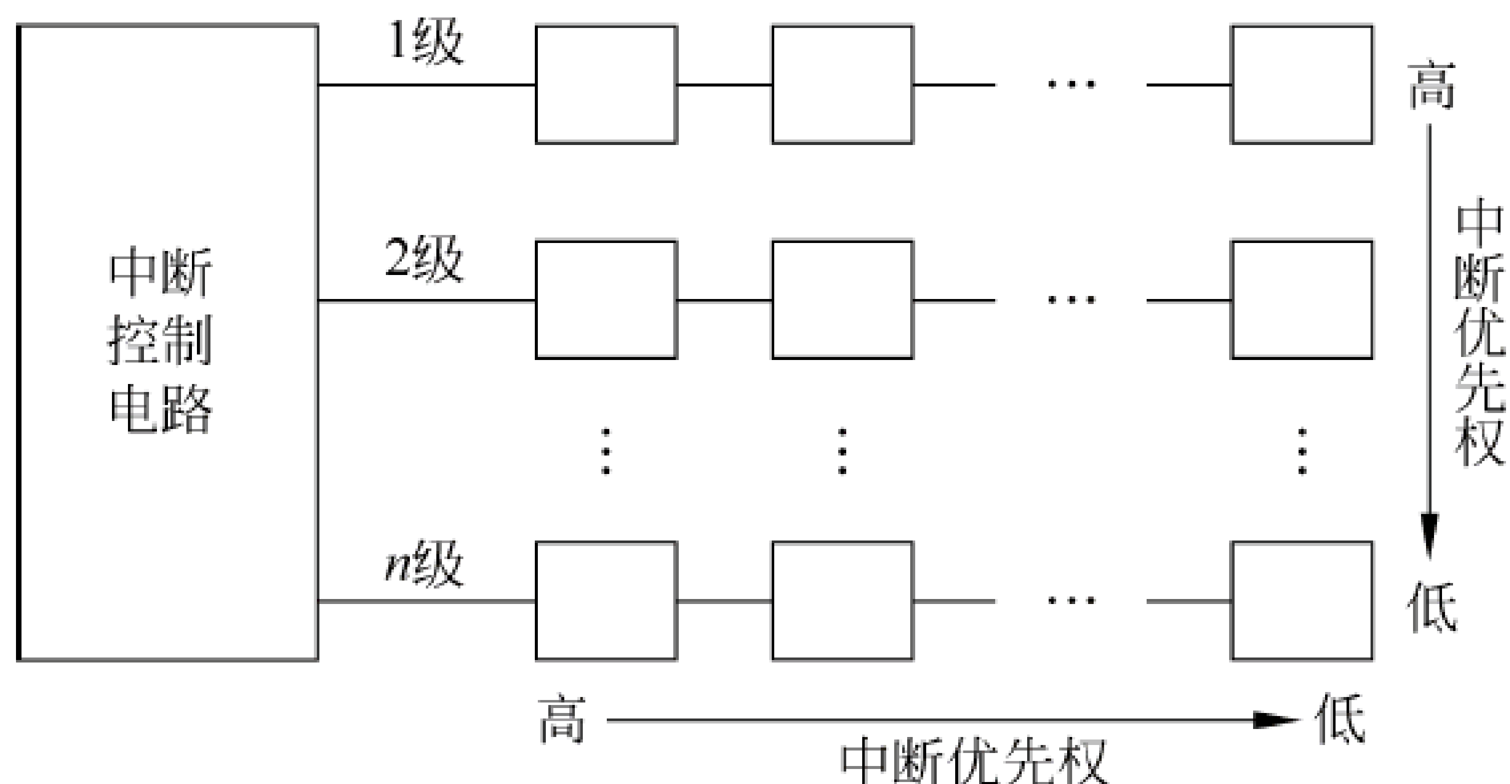


图 5-17　中断优先权和中断优先级

如果一个机器系统只有一个中断级，则称该机器的中断系统为**单级中断系统**；如果一个机器系统有多个中断级，则称该机器的中断系统为**多级中断系统**。

当 CPU 在执行一个主程序时发生了多个中断源的中断请求，中断系统会选择一个优先权高的中断源予以响应。那么，如果 CPU 在执行一个中断服务程序时发生了新的中断，而且新来的中断的优先权更高，中断系统该如何处理呢？有两种解决方案：一是响应，二是不响应，而响应与否取决于机器采用的是单级中断系统还是多级中断系统。

单级中断系统采用的是一种简单的处理方式：当几个不同优先权的中断源同时请求中断时，系统按照它们优先权高低先后顺序一一响应。而当 CPU 正在处理一个中断时，不再响应其他新的中断源的中断请求，即使新的中断源的优先权更高也不予响应，只有一个中断处理完毕后再响应新的中断请求。多级中断系统则允许先处理高优先级的中断源再中断低优先级的中断服务，这称为**多重中断**或**中断嵌套**。理论上多重中断可以无限制地嵌套。

【例 5.1】 设 CPU 在执行一个主程序(非中断服务程序)时，先是 B 设备有了中断请求，经过一段时间后又有了 A 设备和 C 设备的中断请求。

(1) 假设机器系单级中断系统，且 A、B、C 设备的优先权顺序由高到低为 A→B→C。

(2) 假设机器系多级中断系统，且 A、B、C 设备分属不同的优先级，它们的优先级顺序由高到低为 A→B→C。

设 CPU 始终处于开中断状态。试通过图示的方法说明在单级中断系统和多级中断系统中的中断响应和处理过程。

解：

(1) 在单级中断系统中，CPU 首先响应先来的 B 设备的中断请求，处理完成后返回主程序；然后从 A、C 中选择优先权更高的 A 设备的中断请求予以响应，处理完后再次返回主程序；最后响应 C 设备的中断请求，处理完后返回主程序，如图 5-18(a)所示。

(2) 在多级中断系统中,CPU 仍然首先响应 B 设备的中断请求。在 B 设备的中断服务过程中又发生了 A、C 中断,由于 A 设备的中断优先级更高,于是再次中断 B 的中断服务,响应 A 中断请求,待 A 中断处理完后返回 B 的中断服务程序。由于 C 中断优先级比 B 低,因此 B 中断服务不再响应 C 中断,直到 B 中断处理完成后返回到主程序再响应 C 中断请求,如图 5-18(b)所示。

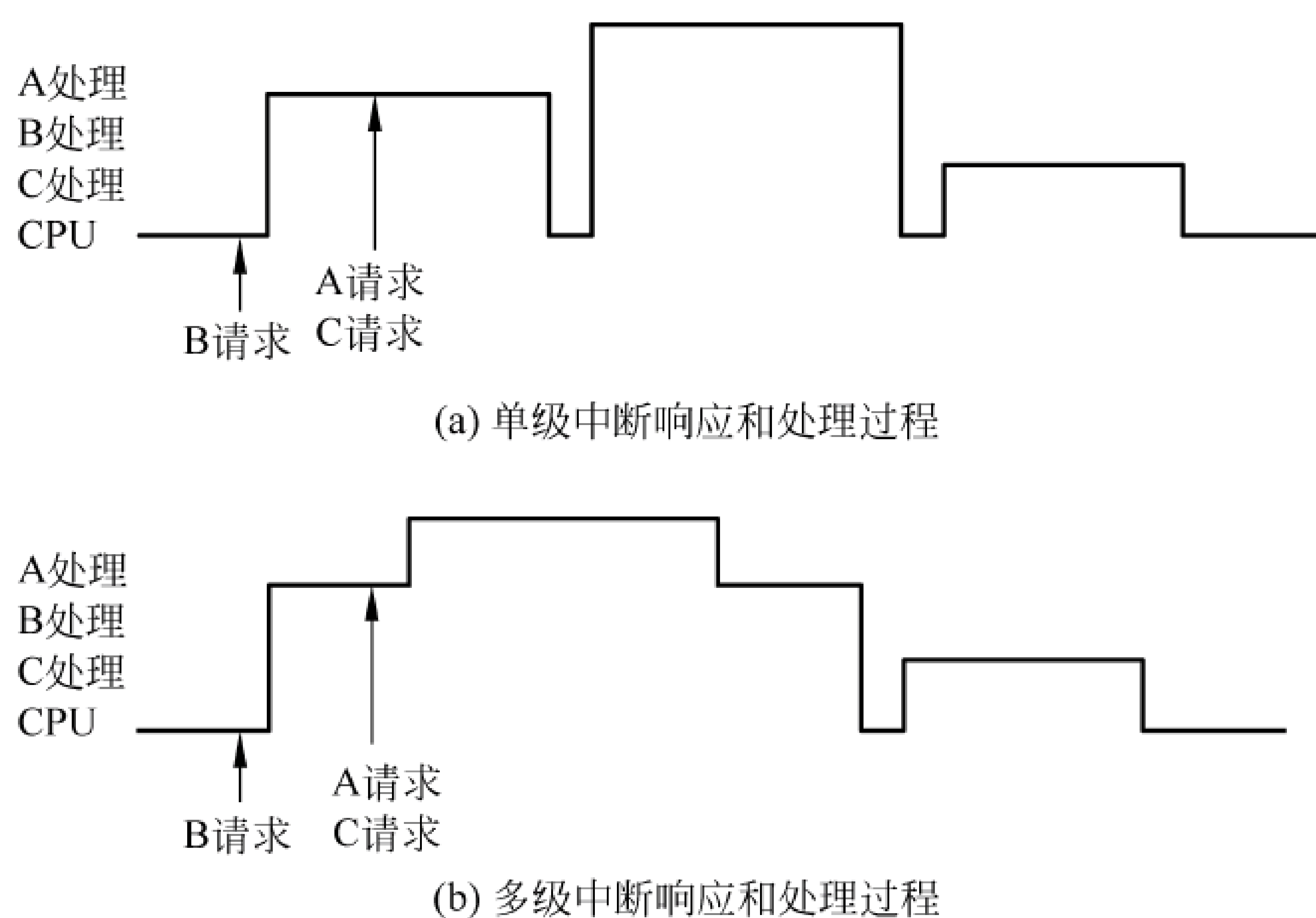


图 5-18 单级中断和多级中断系统中断响应和处理过程

从例 5.1 可以看出,中断优先权决定了各中断源的中断响应顺序,而中断优先级决定了中断处理的顺序,先响应的中断不一定先处理完。

在多级中断系统中,利用中断屏蔽码可以改变中断源的中断处理顺序,使机器的中断系统控制更灵活。对于这一点,仍通过一个例子加以说明。

【例 5.2】 设 CPU 在执行一个主程序时,同时发生了 A、B、C 三个设备的中断请求,A、B、C 的中断优先级顺序由高到低为 A→B→C。

(1) 若 A、B、C 三个设备在执行中断服务程序时的中断屏蔽码如表 5-2 所示,其中“1”表示屏蔽,“0”表示开放,试通过图示的方法说明对 A、B、C 三个设备的中断响应和处理过程。

表 5-2 屏蔽码设置情况 1

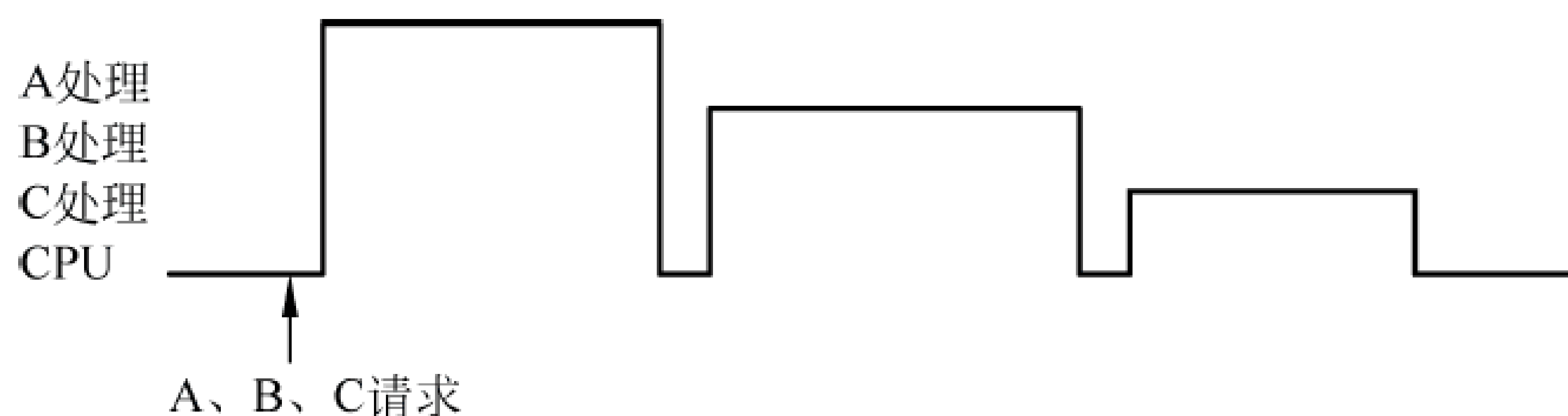
设备名称	屏蔽码		
	A	B	C
A	1	1	1
B	0	1	1
C	0	0	1
CPU	0	0	0

(2) 若 A、B、C 三个设备在执行中断服务程序时的中断屏蔽码如表 5-3 所示,试通过图示的方法说明对 A、B、C 三个设备的中断响应和处理过程。

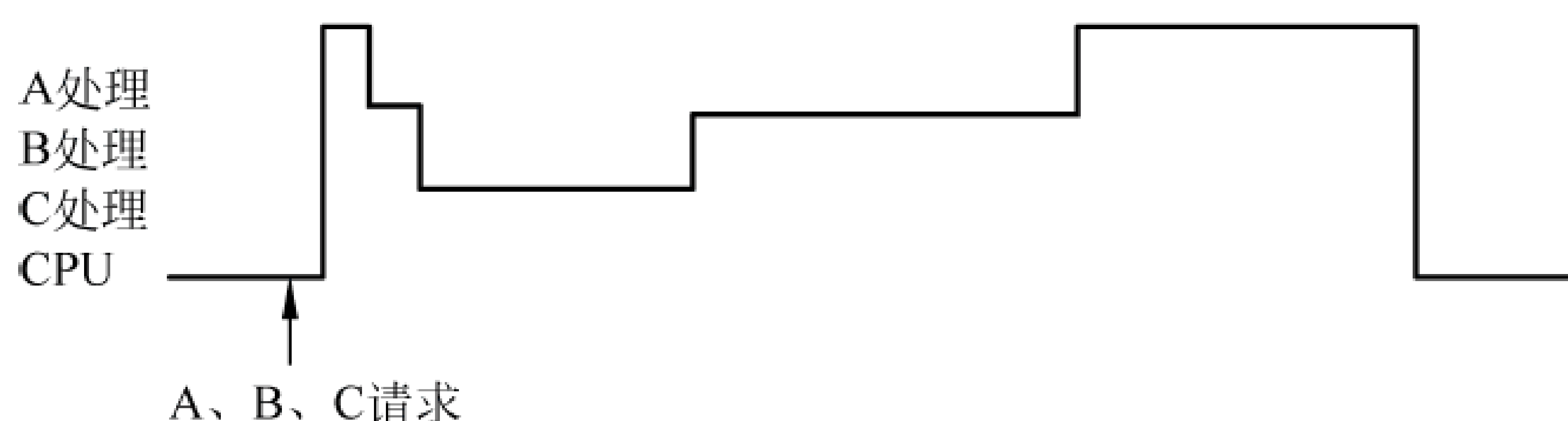
表 5-3 屏蔽码设置情况 2

设备名称	屏蔽码		
	A	B	C
A	1	0	0
B	1	1	0
C	1	1	1
CPU	0	0	0

解：以上(1)、(2)两种情况下的中断响应和中断处理顺序分别如图 5-19 所示。



(a) 表5-2的中断响应和中断处理顺序



(b) 表5-3的中断响应和中断处理顺序

图 5-19 中断屏蔽码改变中断处理顺序

从例 5.2 可以看出，通过改变中断屏蔽码可以改变中断源的中断处理顺序，中断优先级低的中断源可以比中断优先级高的中断源先处理完成。

5. 中断接口电路

在 CPU 与设备之间采用中断控制方式进行传输的中断接口电路中，主要包含数据缓冲寄存器(DBR)、地址译码电路、中断请求触发器(IR)、中断屏蔽触发器(IM)、中断判优排队电路、中断向量逻辑和控制逻辑等，如图 5-20 所示。

根据图 5-20，CPU 控制设备进行一次输出的过程如下：

(1) 首先 CPU 启动设备通过中断接口的控制逻辑向设备发出一个启动控制信号。

(2) 当设备准备好时，向接口发回一个 READY 信号，将接口中的中断请求触发器置 1；若此时设备处于中断开放状态，即接口中的中断屏蔽触发器为 1，则可以通过控制逻辑向 CPU 发出一个中断请求信号 INTR。

(3) CPU 接收到中断请求后，若内部中断允许触发器处于开中断状态，则进入中断响应周期，向接口发出一个中断响应信号 INTA。在该信号控制设备接口进行中断判优和中断源识别。

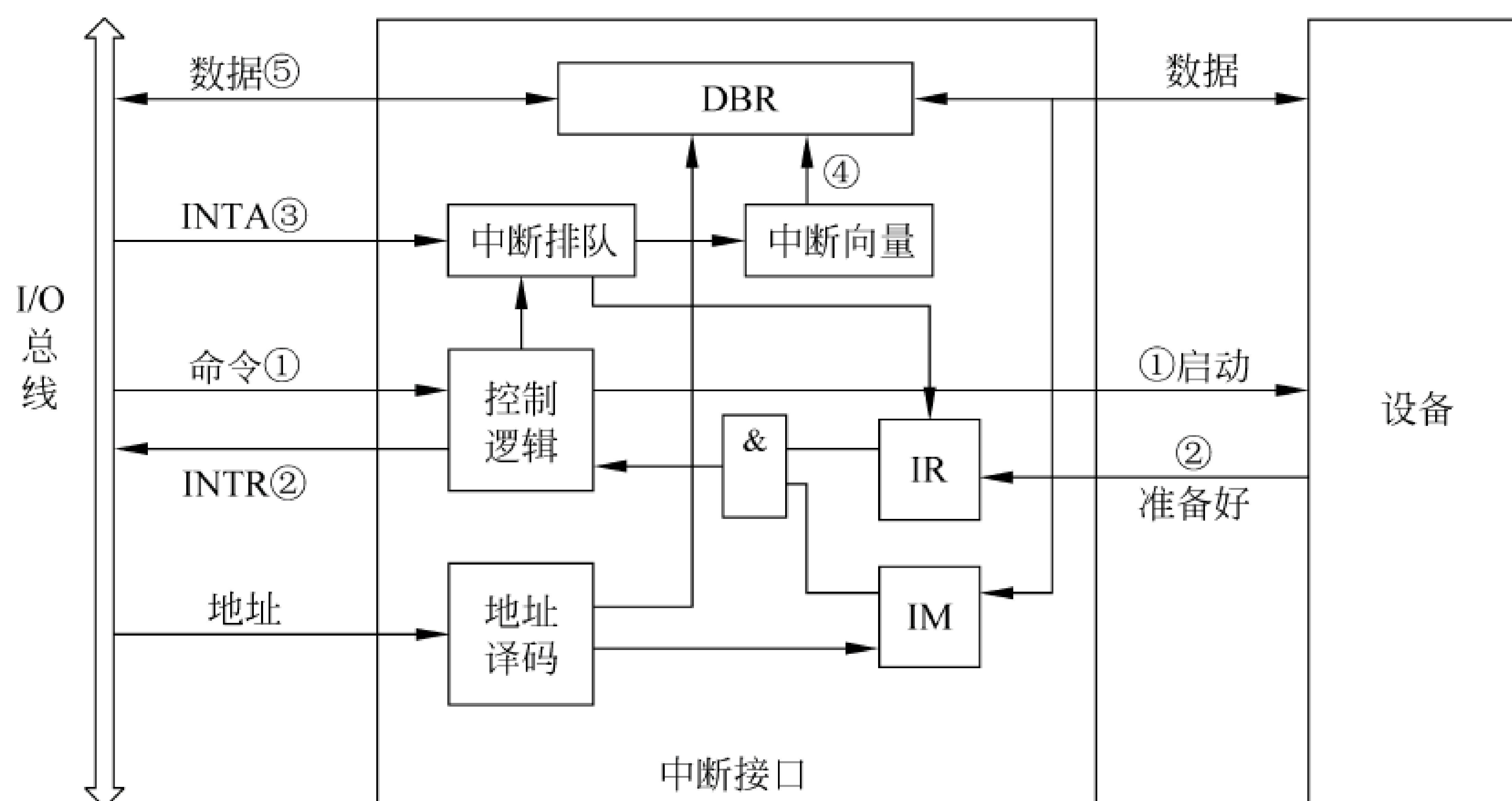


图 5-20 中断接口电路

(4) 若本设备被选中,则将其对应的中断向量号送数据缓冲器,并经数据总线传送给CPU,CPU依据此中断向量找到中断服务程序入口地址,从而转入中断服务程序执行。

(5) 在中断服务程序中,CPU向设备传送一个数据。

至此,CPU通过中断方式向设备传输一个数据的过程就完成了。之后,设备进行输出数据的处理工作,与此同时,CPU返回原主程序继续执行。若设备输出数据处理完毕,又通过接口向CPU发出一个中断请求信号,进入下一次中断响应和中断处理过程。

在计算机中,中断接口电路可以由专门的集成电路芯片实现。例如,Intel 8259A就是一个可编程的中断控制器,其内部结构如图5-21所示。它由8个部分组成:8位中断请求寄存器IRR、8位中断屏蔽寄存器IMR、8位中断服务寄存器ISR、优先级判别器、8位数据总线缓冲器、中断控制逻辑、级联缓冲器/比较器和读写逻辑。

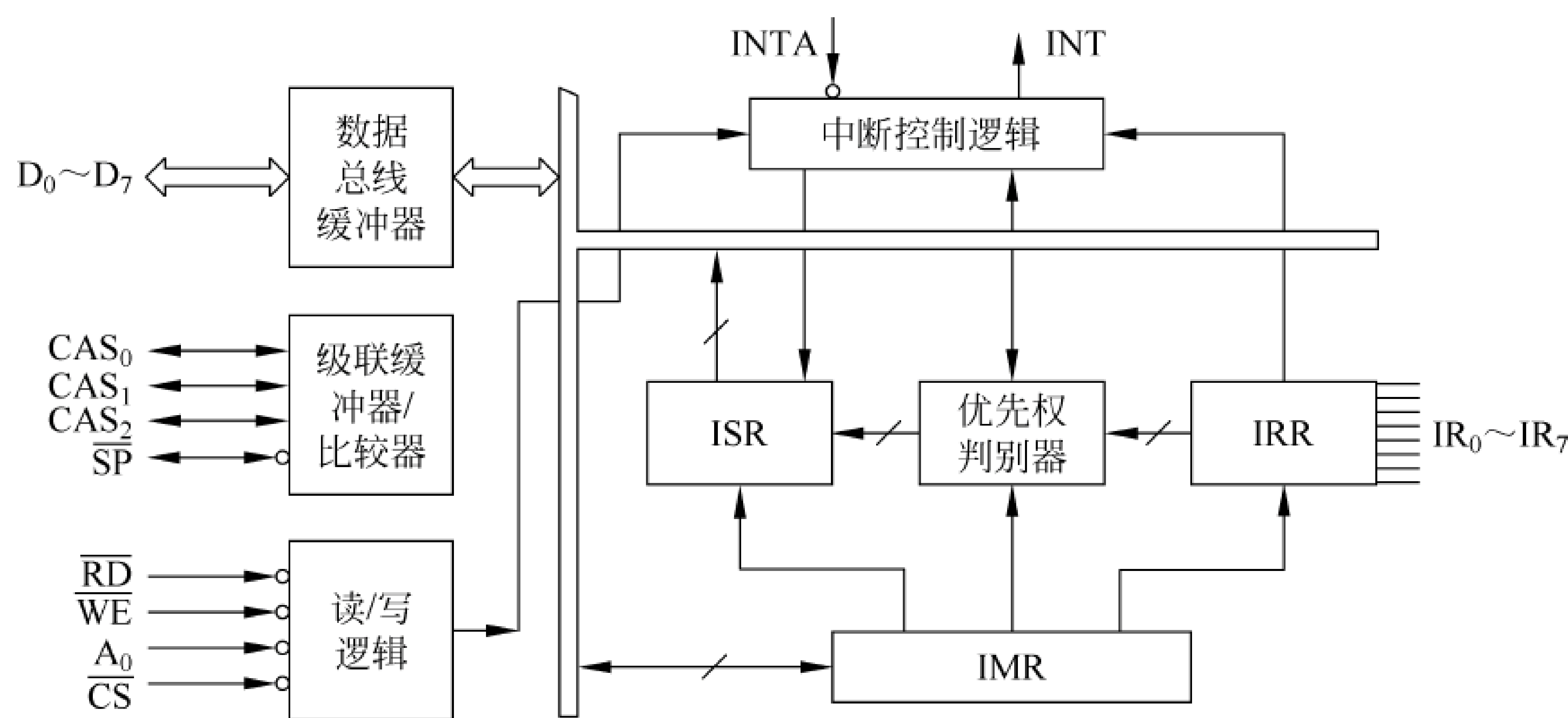


图 5-21 8259A中断控制器

数据总线缓冲器通过数据总线与CPU连接,用于与CPU之间的双向数据交换,如CPU向8259A传送控制字、置中断屏蔽字和8259A向CPU传送中断向量等。读/写逻辑用于CPU对芯片进行读写操作和内部I/O端口的地址译码。一个8259A通过其内部的中

断请求寄存器可以外接8个中断源的中断请求，通过级联控制逻辑，可以将9片8259A级联起来，提供最多64个外部中断源的中断请求和处理。

8259A的中断处理过程如下：

(1) 当外部中断源有了中断请求后，将中断请求寄存器的对应位置位。

(2) 8259A接收这些中断请求，并根据中断屏蔽寄存器的状态将未被屏蔽的所有中断请求送优先级判别器进行判优。

(3) 选择一个优先级最高的中断源，经中断控制逻辑向CPU发出中断请求信号INT。

(4) 若CPU处于开中断状态，则响应中断，向8259A发回中断应答信号INTA，进入由两个总线周期构成的中断响应周期。

(5) 在第一个总线周期，8259A把当前响应的最高优先权中断源所对应的中断服务寄存器的相应位置位，并将中断请求寄存器相应位清零。

(6) 在第二个总线周期，8259A将当前响应的中断源的中断类型号(中断向量号)通过数据总线送至CPU。

(7) CPU依据此中断类型号便可获得该中断源的中断服务程序入口地址，从而转入中断服务程序执行。

8259A提供了4种优先级选择方式：①完全嵌套方式，这是一种固定优先级方式，它规定任何情况下总是IR_0的中断源优先级最高，IR_7的中断源优先级最低。②轮转优先级方式，对每个中断源同等对待，当某个中断源处理完后，把它放到最低级别的位置上，这样保证每个中断源都有机会被响应。③轮转优先级B方式，CPU可以在任何时候规定一个中断源为最高优先级，其他中断源的优先顺序也依据信号线顺序相应确定。④查询方式：通过软件读取中断请求寄存器的状态，依据软件查询的顺序确定中断源的优先级顺序。

8259A提供了两种屏蔽方式：①简单屏蔽方式，8位屏蔽字与8位中断请求位一一对应，为“1”则屏蔽，为“0”则开放。②特殊屏蔽方式，通过屏蔽字的设置，允许低优先权的中断源的中断请求中断高优先权的中断源的中断服务。例如，若8259A被设置为完全嵌套优先级方式和特殊屏蔽方式，则当屏蔽字设置为11001111时，IR_4和IR_5上的中断请求除了可以中断低级别IR_6～IR_7的中断服务外，还可以中断高级别IR_0～IR_3的中断服务。这也就是前面讲到的通过对中断屏蔽字的设置可以改变中断处理的顺序。

8259A的不同控制和工作方式是通过编程来实现的。

6. Pentium CPU的中断机制

Pentium CPU定义了两类中断源，即中断和异常。

中断主要是指外部中断，它是由CPU的外部硬件信号引起的，包括两种情况：一是可屏蔽中断，CPU的INTR引脚上接收到的中断请求，这类中断的允许或禁止受CPU的标志寄存器中的IF位(即中断允许触发器)的控制；二是非屏蔽中断，是由CPU的NMI引脚接收到的中断请求，这类中断不能被禁止。

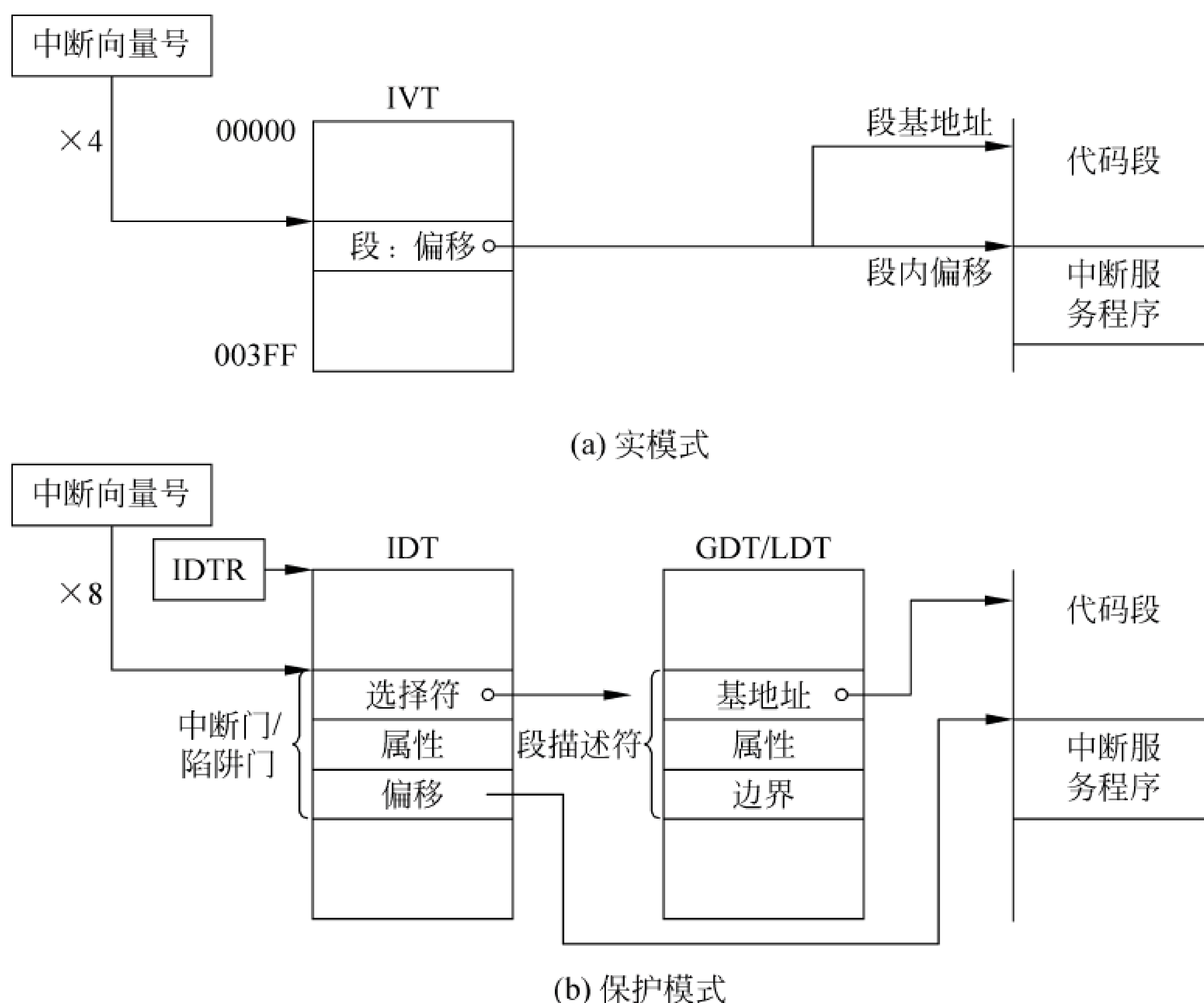
异常主要是指异常中断，它是由指令执行引发的，包括两种情况：一是执行异常，CPU执行一条指令过程中出现错误、故障等不正常情况引发的中断；二是执行软件中断指令(如INT0，INT3，INT*n*等软中断指令)产生的异常中断。

Pentium 共能提供 256 种中断和异常。每种中断源分配一个编号，称为中断向量号或中断类型号。所有的中断和异常按其中断优先级分为 5 级，其中异常中断的优先级高于外部中断的优先级。

Pentium 是采用中断向量法进行中断响应的，即首先得到中断向量号，再通过中断向量号找到中断向量，从而获取中断服务程序入口地址。Pentium CPU 取得中断向量号的方式根据中断源类型的不同也有所不同。对于软件中断指令 INT*n* 来说，其中的 *n* 即为中断向量号；对于错误、故障等异常来说，系统根据异常产生的条件自动指定向量号，每种异常的向量号是固定的，如中断向量号 0 对应除 0 错误异常、中断向量号 4 对应算术溢出错误异常等；而外部中断的中断向量号是系统设计时在中断控制器中预先设定的，当响应中断时由中断控制器提供；非屏蔽中断的中断向量号固定为 2。

所有中断服务程序入口地址信息均存于一个中断向量号检索表中，在 Pentium 实模式下，该表为中断向量表 IVT，保护模式下为中断描述符表 IDT。

实模式：中断向量表 IVT 位于内存地址 0 开始的 1KB 空间。实模式是 16 位寻址，中断服务子程序入口地址（段，偏移）的段寄存器和段内偏移量各为 16 位。它们直接登记在 IVT 表中，每个中断向量号对应一个中断服务程序入口地址。每个入口地址占 4 字节。256 个中断向量号共占 1KB。CPU 取得向量号后自动乘以 4，作为访问 IVT 的偏移，读取 IVT 相应表项，将段地址和偏移量设置到 CS 和 IP 寄存器，从而进入相应的中断服务程序。实模式下中断响应过程如图 5-22(a)所示。



(a) 实模式

(b) 保护模式

图 5-22　Pentium CPU 中断响应过程

保护模式：保护模式为 32 位寻址。中断描述符表 IDT 每一表项对应一个中断向量号，表项称为中断门描述符、陷阱门描述符。这些门描述符为 8 字节长，共对应 256 个中断向量

号,IDT 表长为 2KB。由中断描述符表寄存器 IDTR 来指示 IDT 的内存地址。以中断向量号乘以 8 作为访问 IDT 的偏移,读取相应的中断门/陷阱门描述符表项。门描述符给出中断服务程序入口地址(段,偏移),其中 32 位偏移量装入 EIP 寄存器,16 位的段值装入 CS 寄存器。由于此段值是选择符,还必须访问全局描述符表(GDT)或局部描述符表(LDT),才得到段的基地址。保护模式下中断响应过程如图 5-22(b)所示。

进入中断服务程序后,其处理过程如下:

(1) 当中断处理的 CPU 控制权转移涉及特权级改变时,必须把当前的 SS 和 ESP 两个寄存器的内容压入系统堆栈予以保存。

(2) 标志寄存器 EFLAGS 的内容也压入堆栈。

(3) 清除标志触发器 TF 和 IF。

(4) 当前的代码段寄存器 CS 和指令指针 EIP 也压入此堆栈。

(5) 如果中断发生并伴随有错误码,则错误码也压入此堆栈。

(6) 完成上述中断现场保护后,从中断向量号获取的中断服务程序入口地址(段,偏移)分别装入 CS 和 EIP,开始执行中断服务子程序。

(7) 中断服务子程序最后的 IRET 指令使中断返回。保存在堆栈中的中断现场信息被恢复,并由中断点继续执行原程序。

5.2.3 DMA 控制方式

虽然中断控制方式很好地解决了 CPU 与设备间并行工作的问题,尤其是对于慢速设备来说,采用中断控制方式进行数据传输,可以大大提高 CPU 的利用率。但是,在中断控制方式下,CPU 每经历一次中断,都要进行从中断请求信号的建立、中断源识别、中断响应到中断服务等的操作,在中断服务程序里还要执行一系列的诸如保护现场/恢复现场、开中断/关中断等的指令,这些操作和指令的执行花费了不少时间。对于 CPU 与一些高速设备间采用成组数据交换的应用来说,中断控制方式就有些显得力不从心了。为此,人们提出了一种 DMA 传送控制方式。

1. DMA 的基本概念

DMA 的全称是 Direct Memory Access,即直接存储器访问,这是一种完全由硬件(称为 DMA 控制器)控制主机与设备间进行数据交换的输入输出传送控制方式,它通过在主存与设备间建立一条直接通道的方法,来进一步提高 I/O 数据传输效率。

在机器中,依据各部件所处的地位将它们划分为两大类:一类是主设备,一类是从设备。**主设备**是指能够占用系统总线并通过总线对其他从设备进行控制的设备,一般来讲,主设备能够在总线上给出地址和控制等信号,完成对存储器和外围设备等的访问,如 CPU 就是机器系统中的主设备。而从**设备**是指被主设备控制和访问的设备,如存储器及各种外围设备等。前面讲到的程序控制方式和中断控制方式,都是在 CPU 这一主设备的控制下完成存储器与外围设备间的数据交换的。

为实现 DMA 传送,机器系统专门设置了一个主设备——DMA 控制器,由 DMA 控制器代替 CPU 控制完成存储器与外围设备间的数据交换,DMA 控制器可以像 CPU 一样,通过总线向存储器和外围设备给出地址和控制信号,实现对这些设备的访问和控制。

表 5-4 给出了 DMA 与程序控制方式及中断控制方式的比较。

表 5-4 几种输入输出控制方式的比较

输入输出控制方式	输入输出过程控制	数据传输通道	CPU 效率	适 用 场 合
程序控制方式	软件实现	经 CPU	低	单用户单道程序环境
中断控制方式	软、硬件结合实现	经 CPU	一般	多用户多道程序环境下慢速或速度不确定的设备
DMA 控制方式	硬件实现	不经 CPU	高	成组、高速设备

2. DMA 的工作模式

任何时候，机器的系统总线只能被一个主设备所占用。当 CPU 在执行程序时，需要通过总线进行访存操作，如取指令、读/写数据等，而 DMA 控制器在进行 DMA 传输控制时也要使用总线对存储器进行读写操作等，这时就会产生总线的冲突。为此需要一种机制来协调 CPU 与 DMA 控制器对总线的使用，通常有以下三种方式。

(1) 突发方式(Burst Mode)。

突发方式又称块传送方式，在这种方式下，当 CPU 所执行的程序要对一个 DMA 设备进行一块数据的读/写操作时，CPU 首先对 DMA 控制器进行初始化，并发送以下信息：

① 是读还是写。

② 所读/写的存储器缓冲区的首地址。

③ 所读/写的 DMA 设备的地址。

④ 所读/写的字数。

然后，CPU 暂停对部分总线的控制，将总线控制权交给 DMA 控制器，由 DMA 控制器控制完成本次数据块的传送。当 DMA 传送结束时，DMA 控制器向 CPU 发出一个中断请求信号，并将总线控制权重新交回给 CPU。

这种方式的优点是控制简单，适用于数据传输率很高的 I/O 设备实现成组数据的传送。缺点是 DMA 控制器在占用总线进行 DMA 传送时，CPU 基本上处于不工作状态或保持原状态，在一定程度上影响到 CPU 的利用率。

(2) 周期挪用方式(Cycle Stealing)。

周期挪用方式是一种常用的 DMA 工作模式。这种方式是 CPU 对 DMA 控制器完成一次数据块传送的初始化后继续工作，当 DMA 设备准备好一个数据的传送时，向 DMA 控制器发出 DMA 请求，DMA 控制器收到请求后向 CPU 申请占用总线，并在 CPU 响应后挪用若干个总线周期进行一个数据的传送，然后将总线控制权交还给 CPU。如此重复，直到完成一个数据块的传送。

与前一种方式相比，CPU 不需要暂停，而只是在有了 DMA 请求时暂时让出若干个总线的控制权。而且，如果在 DMA 控制器使用总线的这些周期里，CPU 正好不需要使用总线(如使用总线访存等)，则不影响 CPU 的执行，从而提高了 CPU 的利用率。这种方式的缺点是 DMA 控制器每控制传送一个数据都要向 CPU 申请占用总线和交回总线，在一定程度上影响到 DMA 传送的效率。

(3) 透明方式(Transparent Mode)。

透明方式又称为直接存储器访问方式。CPU 典型的指令周期包含若干个总线周期(又称 CPU 周期),如取指令周期、指令译码周期、取操作数周期和指令执行周期等。在整个指令周期里 CPU 并不是始终需要进行总线操作的,如乘法指令的执行周期只涉及 CPU 内部的乘法运算,而且时间也较长。因此,可以利用 CPU 不进行总线操作的一些总线周期进行 DMA 传送,这样 CPU 和 DMA 控制器可以交替使用总线,而且总线使用的交替可以通过不同的总线周期直接分配,无须 CPU 和 DMA 控制器之间的请求、建立和归还过程。例如,将每个指令的指令译码周期固定分配给 DMA 传输之用,当某个指令译码周期有 DMA 请求时,直接进入 DMA 操作。这就是透明方式的基本思想。这种方式对 CPU 和 DMA 来说效率都是最高的,但决定 CPU 何时不使用总线的硬件设计十分复杂,而且随着 CPU 指令预取、指令流水等并行技术的应用,现代大多数 CPU 每个周期都使用总线,因此这种方式通常不被采用。

为提高 DMA 传输效率,还可以对总线的配置进行设计。如图 5-23(a)所示是一种传统的单总线系统,在这种系统中,DMA 控制器和 CPU、存储器、I/O 接口及设备一起连接在一条单总线上。DMA 控制器每控制一个数据的传输都至少要占用两个 CPU 周期,一个用于对设备的读或写,另一个用于对存储器的读或写,这两个 CPU 周期都需要占用总线。

图 5-23(b)是一种在传统的单总线基础上增加一条专用 I/O 总线的结构,在这种系统里,DMA 控制器对存储器的访问需要与 CPU 共享系统总线,而对 DMA 设备的访问则通过 I/O 总线进行,这样可以减少在 DMA 传输过程中因 DMA 控制器过多占用系统总线而对 CPU 产生的影响。

图 5-23 DMA 控制的总线配置

3. DMA 控制器的组成及工作原理

图 5-24 给出了一个 DMA 控制器的内部结构及与 CPU 等部件相联的组成框架图。其中,从 DMA 的内部结构看,它主要包含以下部件:

(1) 地址寄存器。用于存放所交换的数据块在主存的首地址。在 DMA 传送前,由 DMA 初始化程序将数据块在主存中的首地址送到该地址寄存器。在 DMA 传送过程中,每交换一次数据,地址寄存器自动增 1,以指向下一内存单元,直到一批数据传送完毕为止。

(2) 字计数寄存器。用于记录所传送数据的总字数。在 DMA 传送前,同样由 DMA 初始化程序将所传送数据的总字数送到该寄存器。在 DMA 传送过程中,每传送一个字,字计数器自动减 1,直到减为 0,表示该批数据传送完毕,于是通过中断控制逻辑向 CPU 发出 DMA 中断请求信号。

(3) 数据缓冲寄存器。用于暂存每次传送的数据。通常 DMA 控制器与主存之间采用字传送,而与设备之间可能是字节或位传送。因此 DMA 控制器中还可能包括装配或拆卸字信息的硬件逻辑,如数据移位缓冲寄存器、字节计数器等。

(4) DMA 控制逻辑。它用于负责管理 DMA 的传送过程,包括接收设备的 DMA 请求和向 CPU 申请总线的占用及归还等。

(5) 中断控制逻辑。在字计数器计为 0 时,表示一批数据交换完毕,由中断控制逻辑向 CPU 发出中断请求,请求 CPU 进行 DMA 操作的后处理。

(6) 读写控制逻辑。用于 DMA 控制器对存储器和设备进行读或写操作的控制,其功能与 CPU 的读写控制相同。

(7) 控制字寄存器。一些可编程的 DMA 控制器是以集成芯片的形式提供的,通常它们具有多种工作模式或操作方式,在使用前可通过软件编程的方法对其进行初始化设置。

结合图 5-24,可以给出 DMA 传输控制的过程如下:

(1) 在 DMA 数据传输前,首先由 CPU 通过初始化程序对 DMA 控制器进行预设置,包括将所传输数据块在内存的首地址送 DMA 内部的地址寄存器,将所传输数据的字数送 DMA 内部的字计数寄存器等。

(2) DMA 控制器选择一个 DMA 设备开始工作。当被选中的设备准备就绪时(对输入设备来说就是准备好了一个数据,对输出设备来说就是准备好接收),向 DMA 控制器发出一个 DMA 请求信号 DREQ。

(3) DMA 控制器接收到设备请求后,向 CPU 发出 HOLD 信号,申请占用总线。

(4) CPU 通过 HLDA 信号进行总线响应,同时将其引出脚的地址、数据和部分控制线置为浮空状态,即将总线的控制权让出。

(5) DMA 控制器获得总线控制权后,向设备回答一个 DMA 响应信号 DACK,并开始启动一次数据的传输。

(6) DMA 控制器将其地址寄存器的内容输出到地址总线上,并给出读/写控制信号,控制设备与存储器之间的一次数据交换,然后地址寄存器增 1,字计数寄存器减 1。

(7) 重复以上过程,直到字计数寄存器减为零,DMA 控制器向 CPU 发出中断请求,同时结束 DMA 传输,将总线控制权归还 CPU。

(8) CPU 响应 DMA 中断请求,并进行 DMA 传输的后处理操作。

从上述 DMA 传输控制过程可以看出,存储器与设备的数据交换没有经过 CPU,完全是在 DMA 控制器的控制下完成的,地址寄存器的增 1 操作、字计数器的减 1 操作等都是由硬件自动完成的,每传输一个数据也无须像中断方式那样经历很多过程和执行一大堆指令,

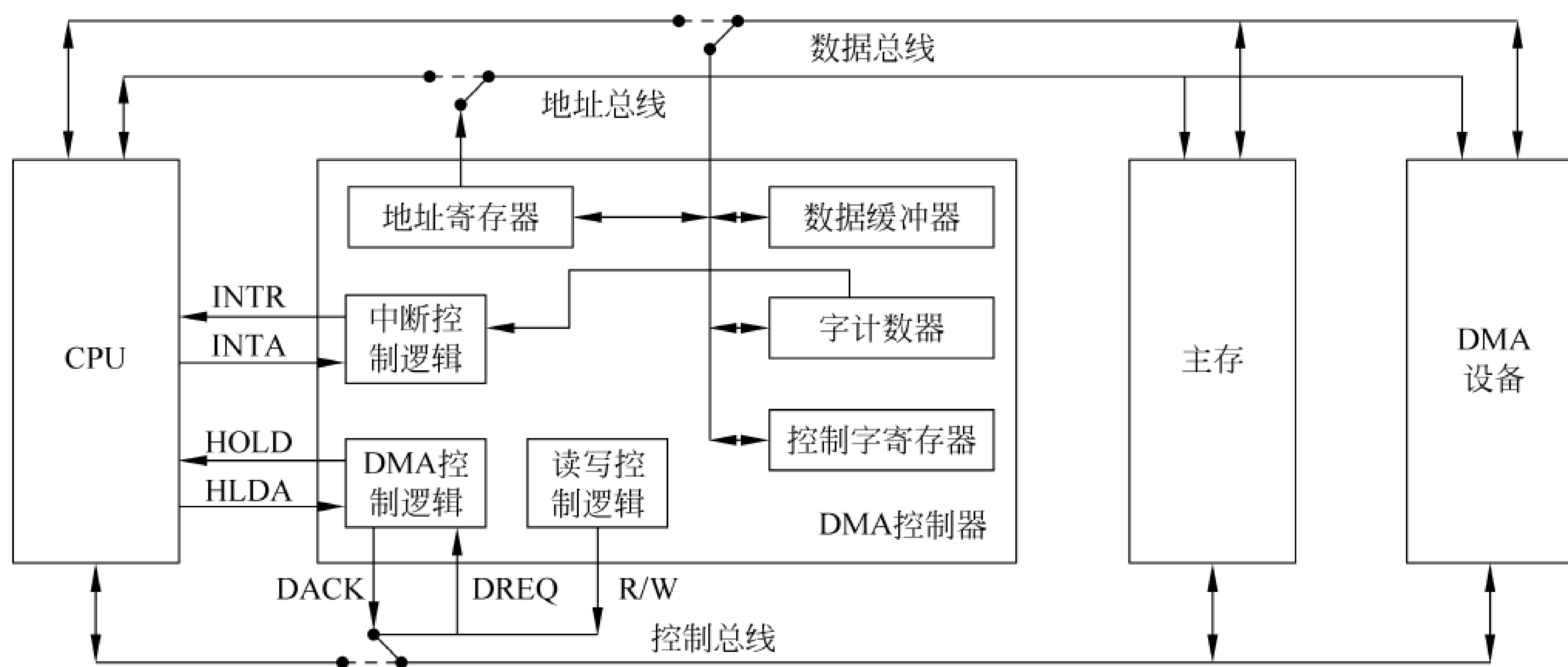


图 5-24 DMA 控制器的组成

因此传输效率大大提高了。

5.2.4 通道控制方式

1. I/O 通道的概念

在前面介绍的几种输入输出传输控制方法中，程序控制方式全部依赖 CPU 进行，中断方式要求 CPU 的不断介入，这两种方式主要适合于对速度要求不高的设备；在 DMA 控制方式中，CPU 无须干预数据传输的过程，而只需进行传输前的初始化和传输后的后处理，由于传输控制是硬件支持，所以适用于高速的数据输入输出。以上几种控制方式广泛应用于微型和小型计算机系统中。但在大型计算机系统中，由于外围设备配置多，输入输出操作频繁，因此对输入输出传输方式有了更高的要求，主要体现在以下两个方面：

（1）尽量减少对 CPU 资源的占用，使用独立的控制部件完成输入输出的操作。

（2）控制更加灵活，输入输出控制部件能适应不同性能的设备的要求。

为此，在 DMA 方式的基础上进一步发展了 I/O 通道方式，以满足大型计算机系统对输入输出控制的要求。

I/O 通道(I/O Channel)又称通道处理器，是一种能执行有限指令集的专用处理器，它通过执行存储在内存中的固定或由 CPU 设置的通道程序来控制设备的输入输出操作。与 DMA 控制器一样，通道也是一个独立的控制部件，但它比 DMA 控制器更进了一步。一方面它是一个处理器，具有有限的指令集，能够执行程序；另一方面它控制灵活，可以适应不同工作方式、不同速度要求和不同数据格式的不同种类的设备的要求。当然，通道处理器还不是一个通用处理器，而是专用于输入输出控制的 I/O 处理器。

通道处理器可以分担 CPU 大部分的 I/O 处理工作，如管理所有低速外围设备的输入输出操作，对 DMA 控制器的初始化工作，控制 DMA 的数据传输、数据格式转换、设备状态检测等，使 CPU 能从烦琐的 I/O 处理中解脱出来，真正发挥其"计算"的能力。

2. I/O 通道的功能

一个具有通道功能的典型机器机构如图 5-25 所示。

一般来讲，通道主要包括寄存器部分和控制部分。寄存器部分包括数据缓冲寄存器、主存地址寄存器、字计数寄存器、通道命令字寄存器、通道状态寄存器等；控制部分包括分时控制、地址分配、数据传输、数据装配和拆卸等控制逻辑。

使用通道方式组织的输入输出系统，一般采用"主机-通道-设备控制器-I/O 设备"四级连接方式。通道对 I/O 设备的控制通过设备控制器或 I/O 接口进行。对于不同的 I/O 设备，设备控制器的结构和功能各有不同，但通道与设备控制器之间一般采用标准 I/O 接口相连接。通道执行指令产生的控制命令经设备控制器的解释转换成对设备操作的控制，设备控制器还能将设备的状态反映给通道和 CPU。

具体来说，通道一般具有以下几方面的功能：

（1）接收来自 CPU 的 I/O 指令，根据指令要求选择设备。

（2）执行 CPU 为通道组织的通道程序，这包括从主存中取出通道指令，对通道指令进行译码，并根据指令的要求向设备控制器发出各种命令。

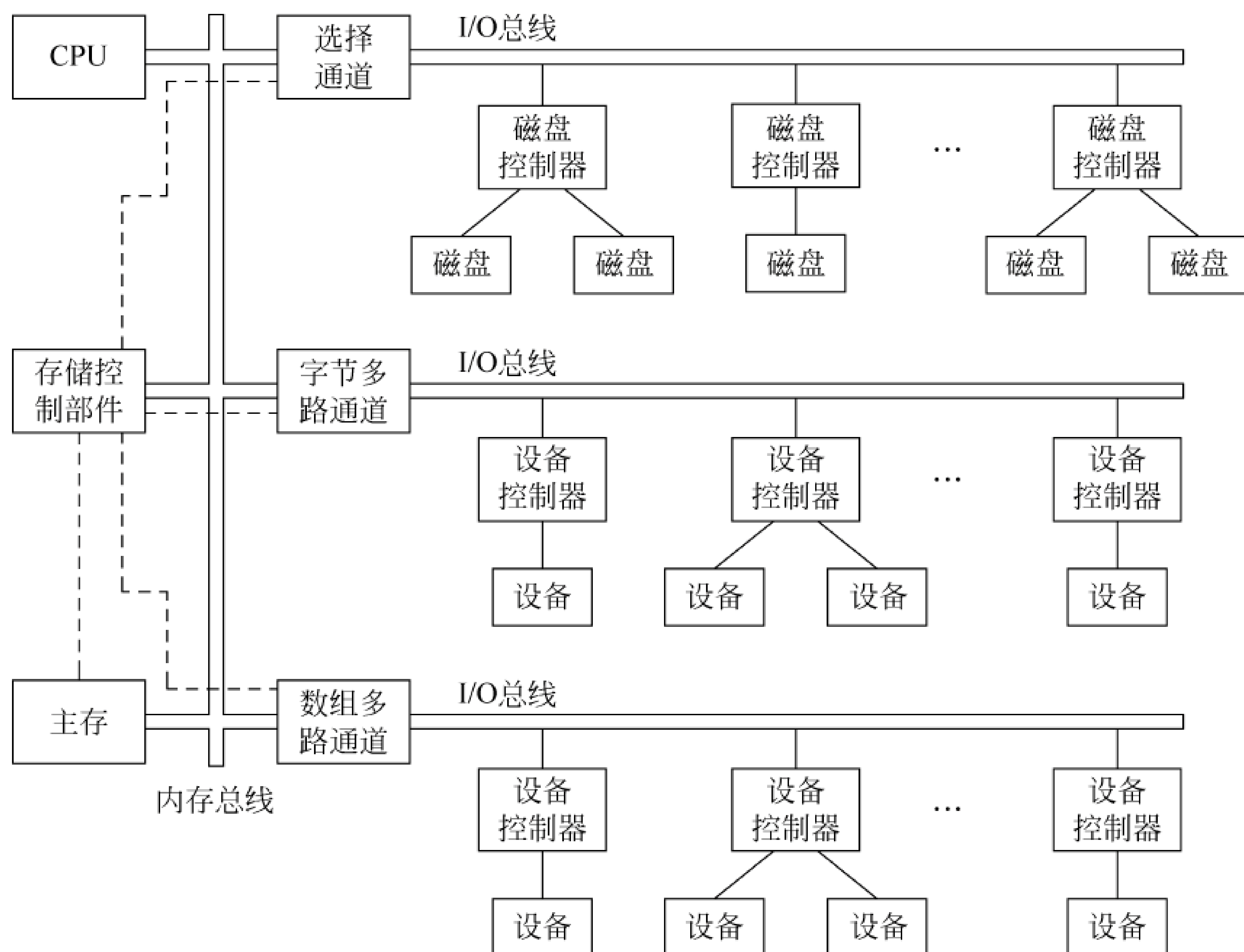


图 5-25 通道控制结构

(3) 控制设备与主存之间的数据传输，提供主存地址和传送的数据字数控制，根据需要完成传输过程中的数据格式转换等。

(4) 检查设备的工作状态，并将完整的设备状态信息送往主存或指定单元保存。

(5) 向 CPU 发出输入输出操作中断请求，将外围设备的中断请求和通道本身的中断请求按次序报告 CPU。

设备控制器的具体任务包括：

(1) 从通道接受通道命令，控制设备完成指定的操作。

(2) 向通道提供设备的状态。

(3) 将各种设备的不同信号转换成通道能够识别的标准信号。

CPU 通过执行 I/O 指令以及处理来自通道的中断，实现对通道的管理。来自通道的中断有两种：一种是数据传输结束中断；另一种是故障中断。通道的管理一般是由操作系统实现的。

3. I/O 通道的种类

按通道的数据传输及工作方式划分，通道可分成字节多路通道、选择通道和数组多路通道三种类型。一个机器系统可以兼有三种通道，也可以只包含其中一种或两种，以适应不同种类设备的需要。

(1) 字节多路通道(Byte Multiplexer Channel)。

字节多路通道用于连接多个慢速或中速的设备，这些设备的数据传送以字节为单位。一般来讲，这些设备每传送一个字节需要较长的等待时间，因此，通道可以以字节为单位轮

流为多个设备服务，以提高通道的利用率。字节多路通道的操作模式有两种：字节交叉模式和猝发模式。在字节交叉模式中，通道将时间分为一个个时间段，有数据传输要求的每一个设备都可以轮流分配得到一个时间段，完成一次与通道间的数据交换。如果某一设备需要传输的数据量比较大，则通道可以采用猝发的工作模式为其服务。在猝发模式下，通道与设备之间的传输一直维持到设备请求的传输完成为止。通道使用一种超时机制判断设备的操作时间（即逻辑连接时间），并决定采用哪一种模式。如果设备请求的逻辑连接时间大于某个额定的值，通道就转换成猝发模式，否则就以字节交叉模式工作。

（2）选择通道（Selector Channel）。

对于高速的设备，如磁盘等，要求较高的数据传输速度。对于这种高速传输，通道难以同时对多个这样的设备进行操作，只能一次对一个设备进行操作，这种通道称为选择通道。选择通道与设备之间的传输一直维持到设备请求的传输完成为止，然后为其他外围设备传输数据。选择通道的数据宽度是可变的，通道中包含一个保存输入输出数据传输所需的参数寄存器。参数寄存器包括存放下一个主存传输数据存放位置的地址和对传输数据计数的寄存器。选择通道的输入输出操作启动之后，该通道就专门用于该设备的数据传输直到操作完成。

（3）数组多路通道（Block Multiplexer Channel）。

数组多路通道以数组（数据块）为单位在若干高速传输操作之间进行交叉复用，这样可减少外围设备申请使用通道时的等待时间。数组多路通道适用于高速外围设备，这些设备的数据传输以块为单位。通道用块交叉的方法，轮流为多个外设服务。当同时为多台外围设备传送数据时，每传送完一块数据后选择下一个外围设备进行数据传送，使多路传输并行进行。数组多路通道既保留了选择通道高速传输的优点，又能同时为多个设备提供服务，使通道的功能得到有效发挥，因此数组多路通道在实际系统中得到较多的应用。特别是对于磁盘和磁带等一些块设备，它们的数据传输本来就是按块进行的。而在传输操作之前又需要寻找记录的位置，在寻找期间让通道等待是不合理的。数组多路通道可以先向一个设备发出一个寻找的命令，然后在这个设备寻找期间为其他设备服务。在设备寻找完成后才真正建立数据连接，并一直维持到数据传输完毕。因此采用数组多路通道可提高通道的数据传输的吞吐量。

字节多路通道和数组多路通道都是多路通道，在一段时间内可以交替地执行多个设备的通道程序，使这些设备同时工作。但两者也有区别，首先数组多路通道允许多个设备同时工作，但只允许一个设备进行传输型操作，而其他设备进行控制型操作；而字节多路通道不仅允许多个设备同时操作，而且允许它们同时进行传输型操作。其次，数组多路通道与设备之间的数据传送的基本单位是数据块，通道必须为一个设备传送完一个数据块以后才能为别的设备传送数据块；而字节多路通道与设备之间的数据传送的基本单位是字节，通道为一个设备传送一个字节之后，又可以为另一个设备传送一个字节，因此各设备与通道之间的数据传送是以字节为单位交替进行的。

4. I/O 通道的工作过程

在使用了通道的机器里，CPU 有两种工作状态：一是目态，二是管态。目态是指 CPU 执行用户的目标程序，而管态是指 CPU 执行管理程序。通道传输过程是由用户程序执行

一条广义访管指令引起的，其输入输出过程如图 5-26 所示。

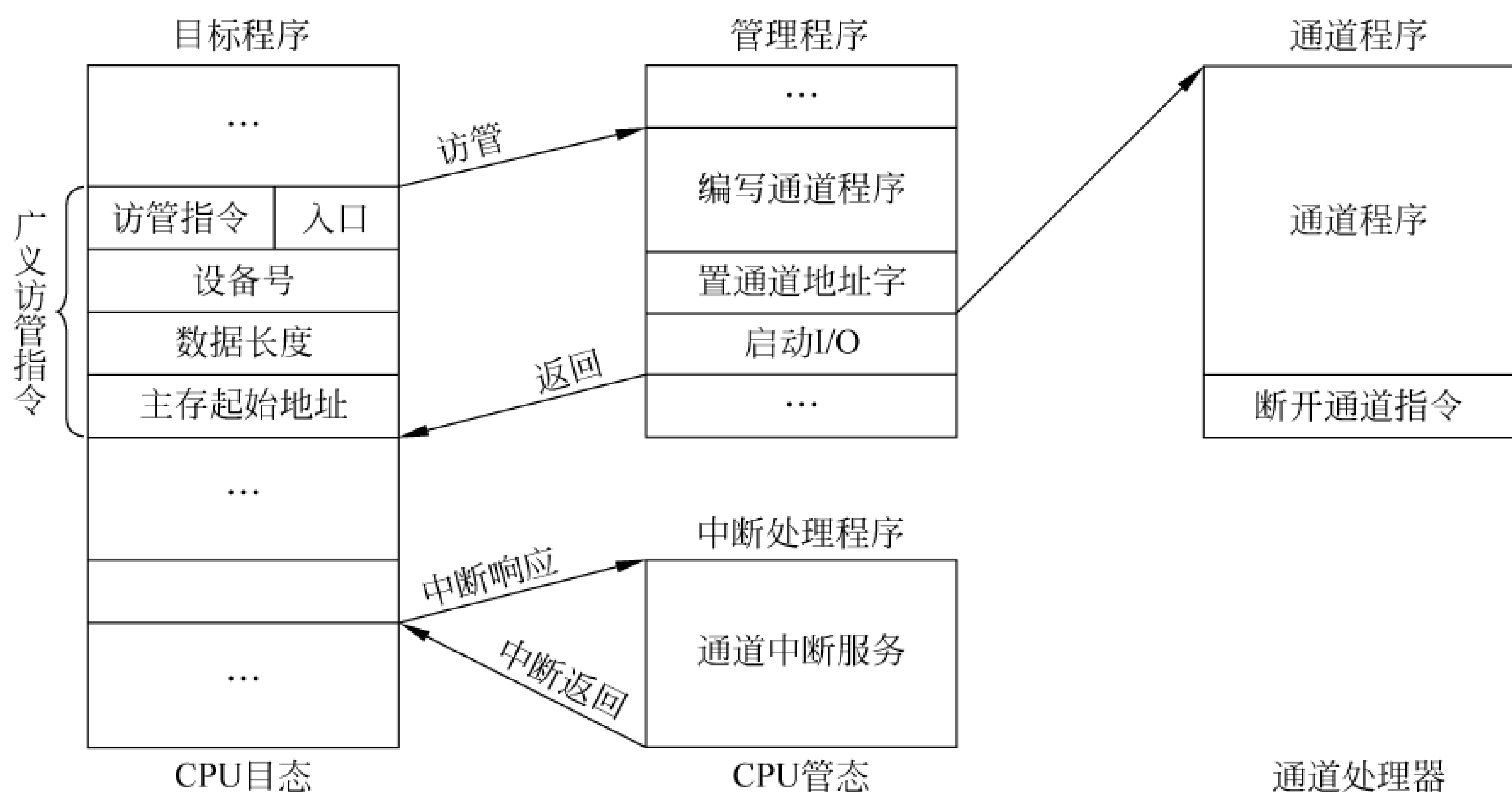


图 5-26　通道数据传输过程

通道完成一次数据传输的主要过程分为以下 4 个步骤：

(1) 用户程序使用访管指令进入管理程序。

(2) CPU 根据用户的访管指令组织形成一个通道程序，并启动通道。

(3) 通道处理器执行 CPU 为它组织的通道程序，控制完成指定的数据输入输出传输操作。

(4) 通道在执行完通道程序后向 CPU 发出中断请求。

CPU 执行用户程序和管理程序以及通道执行通道程序的时间关系如图 5-27 所示。

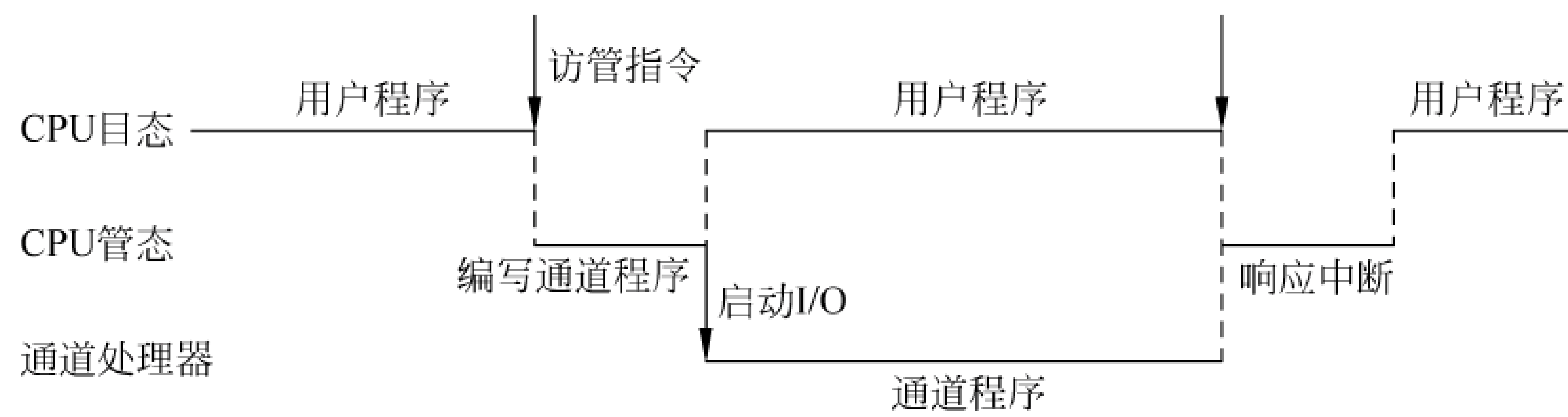


图 5-27　CPU 和通道执行程序时间关系

由于大多数计算机的 I/O 指令都属于管态指令，因此，用户通过在目标程序中设置一条要求进行输入输出的广义访管指令来使用设备。广义指令由访管指令和一组参数组成，它的操作码实际上就是对应此广义指令的管理程序入口。访管指令是目态指令，当目标程序执行到要求输入输出的访管指令时，产生一个自愿访管中断。CPU 响应中断后，转向该管理程序入口，进入管态。

管理程序根据用户程序中广义指令提供的参数，如设备号、交换数据长度和数据在主存中的起始地址等信息来编制形成一段通道程序。通道程序由若干条通道指令组成，能够完成 CPU 一条 I/O 指令所要求的操作。通道程序编制好后，放在主存中与这个通道相对应的通道程序缓冲区中。另外，在管理程序中还要把通道程序的入口地址放入相应的通道地

址单元。在管理程序的最后，用一条启动 I/O 设备的指令来启动通道开始工作。

接下来，通道处理器开始执行 CPU 为它组织的通道程序，完成指定的输入输出传输工作。此时，CPU 也返回用户程序继续执行，通道处理器和 CPU 并行工作。当通道处理器执行完通道程序后，执行一条断开通道指令，向 CPU 发出中断请求，CPU 响应中断并进行相应的后处理。

至此，一次由通道控制的主存与设备间的数据交换结束。

5. 通道结构的发展

通道结构的进一步发展，出现了两种计算机 I/O 体系结构：

(1) 输入输出处理器(IOP)。IOP 是通道结构的 I/O 处理器，它可以和 CPU 并行工作，提供高速的 DMA 处理能力，实现数据的高速传送。但是它不是独立于 CPU 工作的，而是主机的一个部件。有些 IOP，如 Intel 8089 IOP，还提供数据的变换、搜索以及字装配/拆卸能力。这类 IOP 广泛应用于中小型及微型计算机中。

(2) 外围处理机(PPU)。PPU 基本上是独立于主机工作的，它有自己的指令系统，完成算术/逻辑运算、读/写主存储器、与外围设备交换信息等。有的外围处理机直接就选用已有的通用计算机。外围处理机方式一般应用于大型计算机系统中，用于处理大量而繁杂的输入输出操作，以使 CPU 能从输入输出控制操作中最大限度地解脱出来，专用于“计算”的处理。

5.3 外部存储器的组织

在第 4 章讲到，存储器按在计算机中所起的作用分为两大类，即内存和外存。内存又称为主存，主要由半导体存储器件构成，其特点是速度快；而外存又称为辅存，从现代存储介质的发展看，主要由磁介质存储器和光盘存储器构成，其特点是容量大，价格低廉。在计算机中，外存是作为设备来进行管理的。5.3 节主要讲述构成计算机外存的磁介质存储器和光盘存储器。

5.3.1 磁盘存储器

磁盘是在一定的基质上涂上一层磁性材料而构成的圆盘，在磁盘表面利用磁存储原理来存储信息。磁盘分为硬盘和软盘两种。硬盘和软盘因它们的基质分别使用了“硬”的金属合金和“软”的聚酯薄膜而得名。硬盘和软盘在构成上以及容量和访问速度上有所不同，但从信息的存储原理来讲，它们是完全相同的。

1. 磁记录原理和读写方式

磁盘是依靠由一个个同心圆组成的磁道上的具有不同磁化方向的磁化元来存储 0、1 信息的。对这些磁化元的读写是通过一个磁头来进行的。磁头是由磁性材料制作而成的，形状如同一个矩形环，靠近磁道方向上开有一个小间隙，在磁头上还分别绕有一组写线圈和读线圈。磁头在对一个磁道进行读写操作时，磁头固定不动，而磁道运动，如图 5-28 所示。

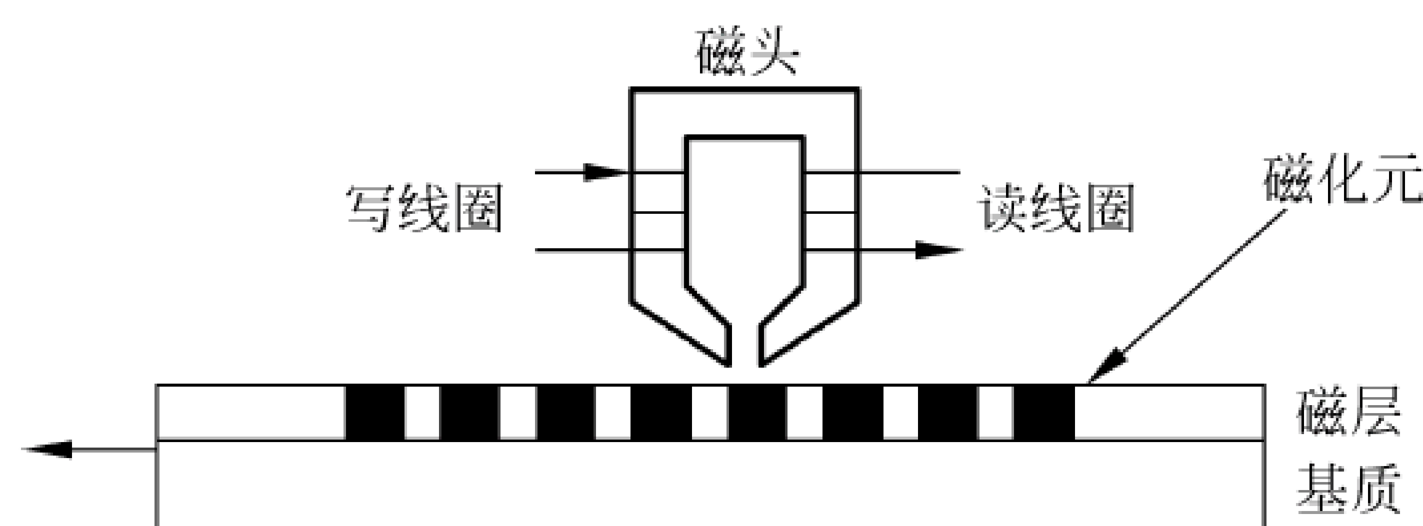


图 5-28 磁盘读写原理

进行写操作时，在写线圈中通过一定方向的脉冲电流时，磁铁芯内就产生一定方向的磁通，并在磁头间隙处产生很强的磁场。在这个磁场作用下，位于磁头下的磁道上的某个固定单元就被磁化成相应极性的磁化元。每个磁化元记录一位二进制位，当磁盘相对于磁头运动时，就可以连续写入一连串的二进制信息。

进行读操作时，经过磁头的磁化元会使磁铁芯内产生磁通的变化，从而使读线圈中产生一定的感应电势，经转换变成一定方向的脉冲电流，由此脉冲电流即可判定所读出的是 0 还是 1。

传统的磁盘是将读和写操作共用一个磁头完成，而新一代的硬盘系统则采用了不同的读写机制，它将读写操作分别在两个不同的磁头上完成，写机制与传统方式相同，而读机制则有所不同。读磁头是由一个磁阻（Magneto Resistive，MR）式感应器组成的，MR 的电阻大小取决于在它下面运动的介质的磁化方向。让电流通过 MR 感应器，此时电阻的变化作为电压信号被检测出来，从而检测出读出的是 0 还是 1。采用这种新的读机制，允许有更高的读操作频度，从而使磁盘可以达到更高的存储密度和读操作速度。

2. 磁盘存储器的物理构成

磁盘存储器主要由磁盘片、磁盘驱动器和磁盘控制器等组成。软盘是由单个盘片构成的，而硬盘则由多个盘片构成，通常称为盘片组。

磁盘驱动器由机械和电器两大部分组成。硬磁盘中，盘片组固定安装在一个主轴上，主轴通过传动带与一个主电机相连。硬盘组的每一个盘面上都有一个独立的磁头，磁头通过磁头臂连接在一个固定滑轨上运动的小车上，再通过小车与一个定位驱动装置相连，如图 5-29 所示。

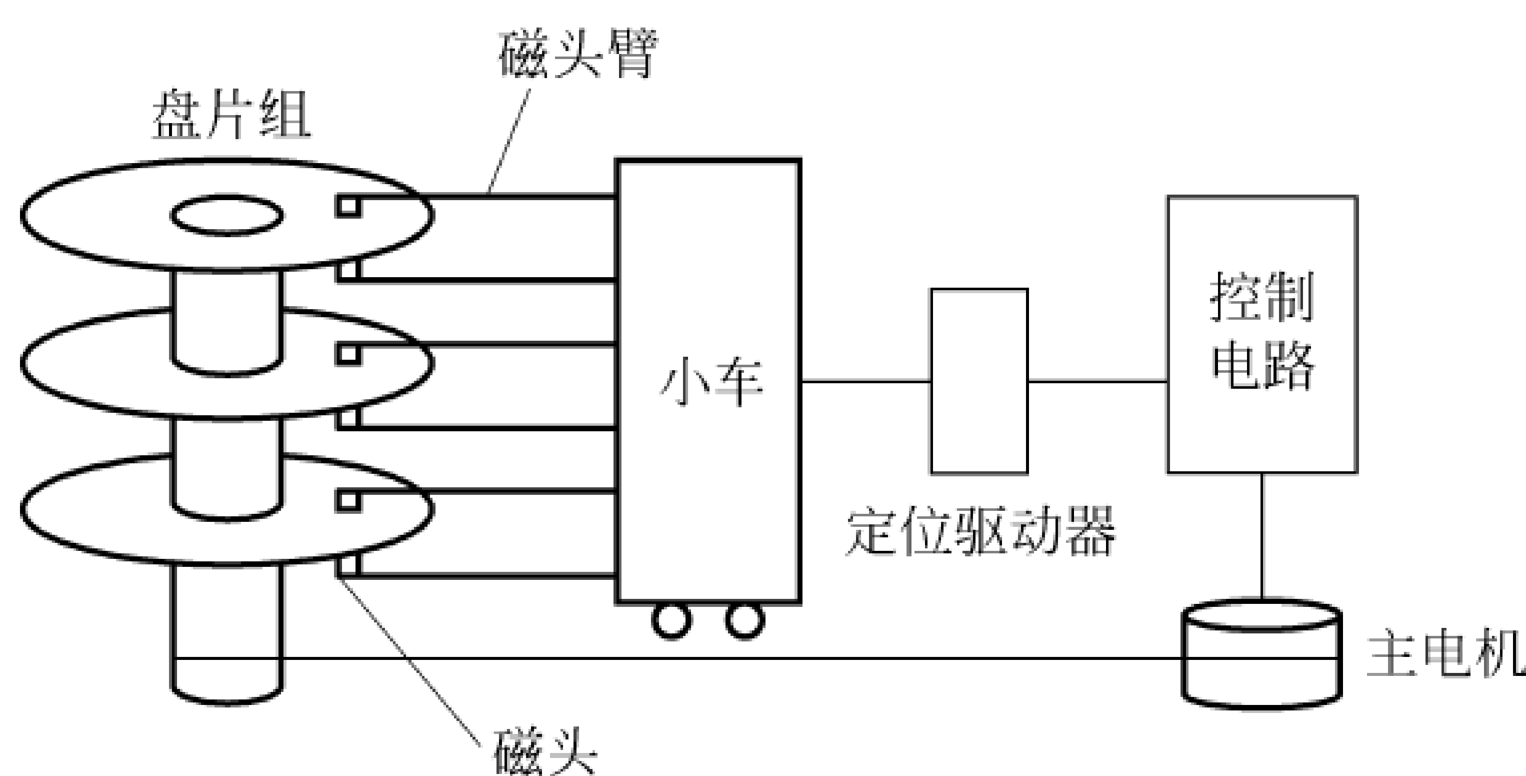


图 5-29 磁盘存储器的物理组成

磁盘驱动器的机械部分一方面驱动盘片组按一定转速绕主轴转动，另一方面驱动小车

通过磁头臂带动磁头沿径向运动,进行磁道的定位。目前磁头小车的驱动方式采用步进电机或音圈电机两种。步进电机靠脉冲信号驱动,控制简单,整个驱动定位系统是开环控制的,因此定位精度较低,一般用于道密度不高的硬磁盘驱动器。而音圈电机是线性电机,可以直接驱动磁头作直线运动,驱动定位系统是一个带有速度和位置信息反馈的闭环控制系统,驱动速度快,定位精度高,因此被更多的硬磁盘驱动器所采用。

磁盘驱动器的电器部分主要是由一些控制电路组成的,其功能主要包括主电机和定位电机的控制,磁头的选择和读写控制,磁盘索引的识别和扇区的定位等。

磁盘控制器是主机与磁盘驱动器之间的接口。磁盘存储器是高速外存设备,它与主机之间采用成组数据交换方式。作为主机与驱动器之间的控制器,磁盘控制器有两个接口:一个是与主机的接口,控制磁盘与主机之间通过外部总线交换数据;另一个是与磁盘驱动器的接口,根据主机命令控制磁盘驱动器的操作。另外,磁盘控制器还能完成磁盘与主机之间的数据格式转换以及数据的编码和解码工作。

3. 磁盘存储器的数据组织

磁盘是以“盘面|磁道|扇区”的方式来进行数据组织的。磁盘的盘面由一个个同心圆环组成,每一个圆环称为一个**磁道**。当对磁盘进行读写时,磁头定位在磁道上,为防止或减少由于磁头定位不准确或磁域间的干扰所引起的错误,相邻磁道间有一定的间隙。一般来讲,一个磁盘面有上千个磁道。

磁道又进一步被分割成几十到上百个等长的圆弧,每一段圆弧称为一个**扇区**。一个扇区可以存储若干位信息,它也是磁盘与主机之间交换信息的基本单位,也就是说,主机对磁盘的读写操作一次至少是一个扇区。一个扇区所存储二进制位数的大小由操作系统对磁盘进行格式化时确定,大多数系统所定义的扇区的大小为512B～2KB。相邻扇区间同样留有一定的间隙。磁盘的数据组织形式如图5-30所示。

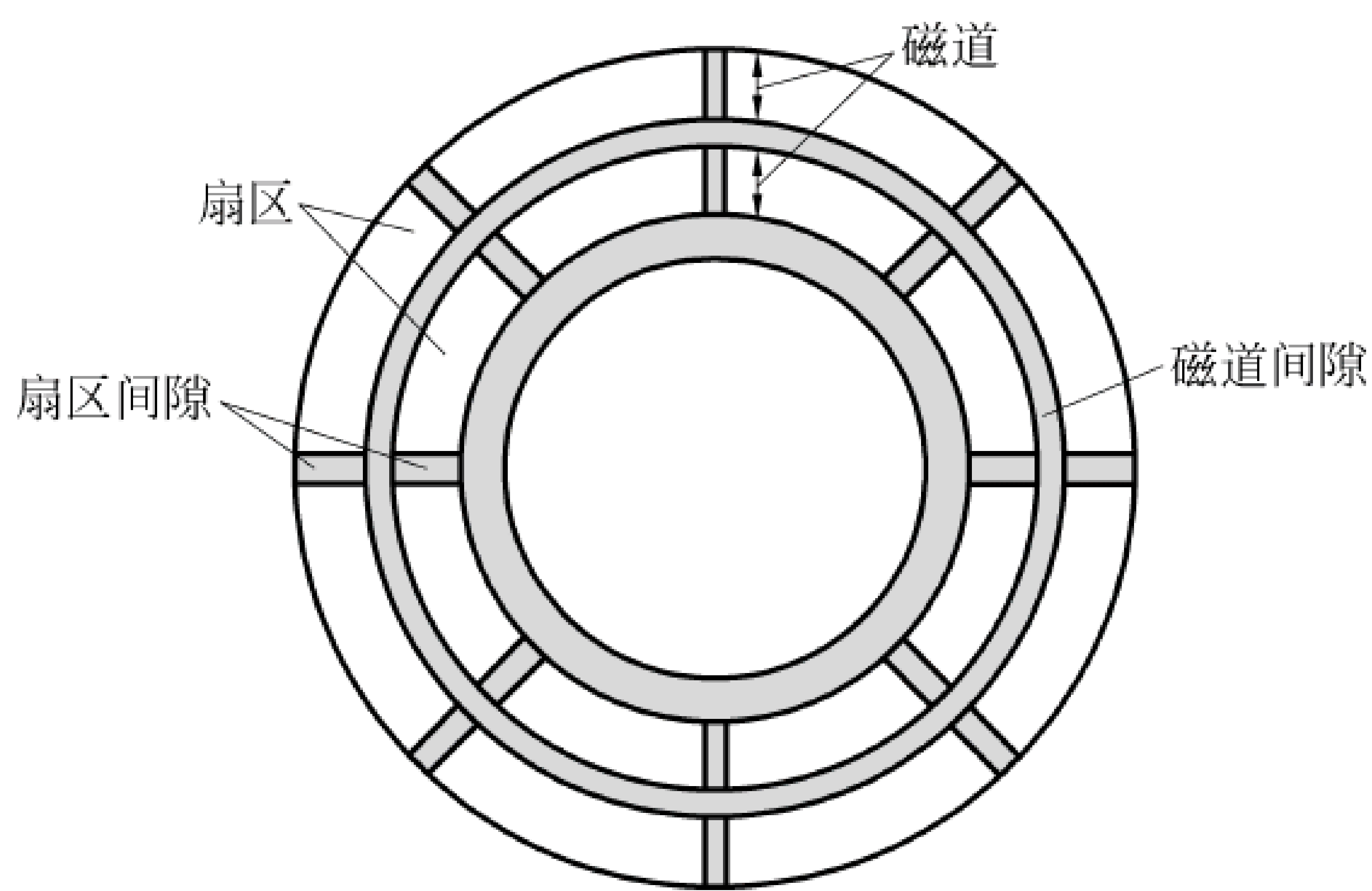


图5-30 磁盘的数据组织分布图

当磁盘转动时,内圈磁道经过磁头的速率要比外圈慢。因此,需要寻找一种方式来补偿速率的变化,使磁头能以同样的速率读写所有的磁道。一种有效的办法是通过增大信息位的间隙来实现。这样以恒定角速度(Constant Angular Velocity,CAV)转动的盘,能使磁头以相同的速率来扫描所有信息位。图5-30就是一种采用CAV技术的磁盘布局格式,它的

好处是，能方便地以磁道号和扇区号来直接寻址各个数据块。为将磁头从当前位置移到指定位置，只需将磁头径向移动到指定磁道，然后等待指定扇区转到磁头下即可。CAV 的缺点是，外圈的长磁道上存储的数据位数与内圈的短磁道相同，浪费了长磁道的存储空间。

一个磁道的存储容量是与磁道的长度（周长）有关的。在上述 CAV 方式中，为使所有磁道的容量相同，从外圈磁道到内圈磁道的线性密度是逐渐增大的，磁盘的容量也就受到最内圈磁道所能达到的最大记录密度的限制。为提高存储容量，现代磁盘系统使用一种称为多区式记录（Multi Zone Recording，MZR）的技术。它将磁盘盘面划分成几个区（典型的是 16 区），每个区中的所有磁道的扇区数是相同的，从而磁道的容量也是相同的；但不同区的磁道所拥有的扇区数是不同的，从而容量也就不同，越往内圈的区，磁道容量越小。当磁头由一个区移动到另一个区时，容量的改变会引起磁头上读写时序的变化，也就使磁盘控制电路的控制更为复杂，但这种电路的复杂所带来的是磁盘容量的提升。

为使磁头能对扇区进行准确定位，磁道上会有一个起始点，称为磁盘索引，同时每个扇区还有起点和终点的标识。磁盘经格式化后，相关控制信息被记录在磁盘上，并由磁盘驱动器所识别和使用。

磁盘格式化的例子如图 5-31 所示。在此例中，每个磁道包含 30 个固定长度扇区，每个扇区有 600B，其中 512B 为有效数据，剩余的为磁盘控制器使用的控制信息。其中各控制信息的含义如下。

（1）ID 域：是唯一的一个标识或地址，用于定位具体扇区。

（2）同步字节：是一个特殊的位序列，用来定义区域的起始点。

（3）磁道号：用来标识磁盘上的一个磁道。

（4）磁头号：用来标识磁盘组中的一个磁头。

（5）扇区号：用来标识磁道上的一个扇区。

（6）CRC：循环冗余校验码。

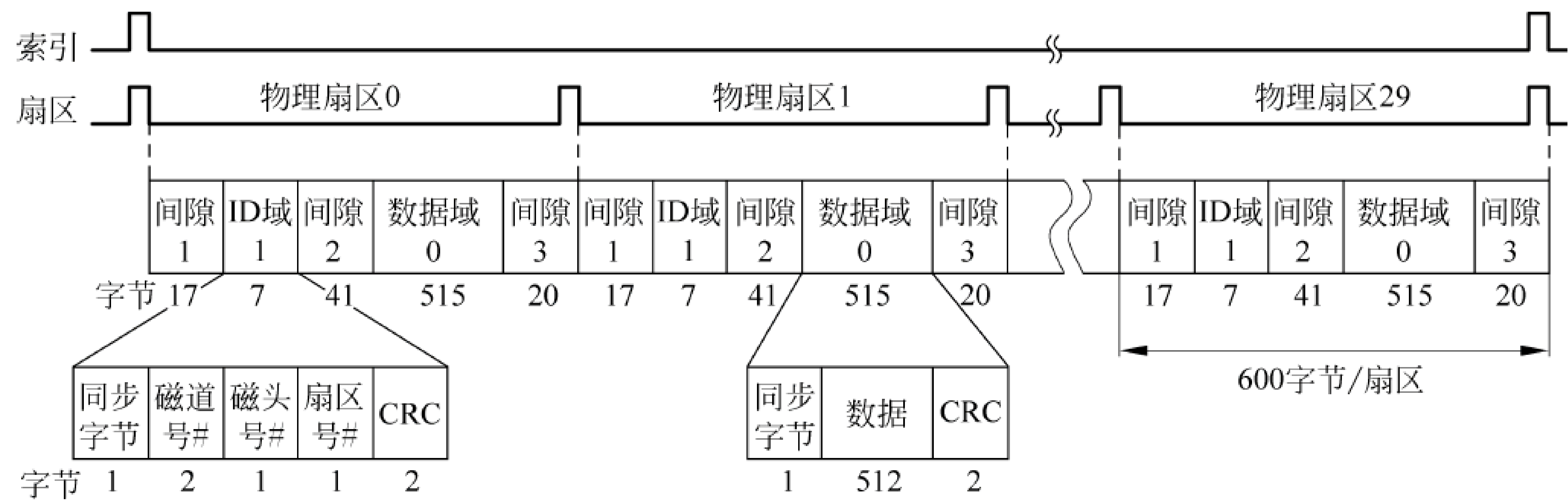


图 5-31　磁盘格式化形式

4．磁盘存储器的性能参数

磁盘存储器的主要性能指标包括存储密度、存储容量和访问时间等。

1）存储密度

磁盘表面的存储密度主要分为道密度和位密度。**道密度**是指沿磁盘径向单位长度上的

磁道数，单位为道/英寸(TPI)或道/毫米(TPM)。**位密度**是指磁道单位长度上能记录的二进制代码位数，单位为位/英寸(BPI)或位/毫米(BPM)。

2）存储容量

一个磁盘存储器所能存储的字节总数，称为磁盘存储器的存储容量。存储容量有格式化容量和非格式化容量之分。**格式化容量**是指按照某种特定的记录格式所能存储信息的总量，也就是用户可以真正使用的容量。**非格式化容量**是磁记录表面可以利用的磁化单元总数。将磁盘存储器用于某计算机系统中，必须首先进行格式化操作，然后才能供用户记录信息。从图 5-31 可以看出，经格式化后的磁盘的每个扇区内部和扇区之间都留有间隙及控制和校验信息，因此，格式化容量比非格式化容量小，一般为非格式化容量的 60%～70%。

在所有磁道容量相同的磁盘存储器中，格式化容量可以按下述方法计算：

磁盘存储器总容量＝盘面数×每面容量

面容量＝磁道数×每道容量

道容量＝扇区数×每扇区容量

而非格式化容量则可以按下述方法计算：

磁盘存储器总容量＝盘面数×每面容量

面容量＝磁道数×每道容量

磁道数＝道密度×径向有效距离

道容量＝位密度×磁道周长

3）访问时间

磁盘存储器的访问时间主要由寻道时间、旋转延时和传送时间三部分组成。

当磁盘驱动器操作时，磁盘主轴电机带动盘片以恒定的速度转动。为了读或写，磁头必须精确定位在所含数据的磁道和扇区的起始处。磁道选择包括在可移动磁头系统中移动磁头或在固定磁头系统选择某个磁头。在可移动磁头系统中，磁头定位到该磁道所花的时间称为**寻道时间**。无论哪一种磁头系统，一旦磁道选定，磁盘控制器将处于等待状态，直到相关扇区旋转到磁头可读写的位置，这段时间称为**旋转延时**。寻道时间和旋转延时的总和称为存取时间，即定位到读写位置的时间。从所访问的扇区头开始，整个扇区从磁头下经过，即完成了该扇区的数据传送，这部分时间称为**传送时间**。

寻道时间：由于寻道时间是不确定的，因此一个磁盘存储器的寻道时间一般取平均寻道时间。平均寻道时间是最大寻道时间与最小寻道时间的平均值，目前硬盘的平均寻道时间通常在 5ms 到 10ms 之间，而 SCSI 硬盘则应小于或等于 8ms。

旋转延时：每次对一个磁道上某个扇区的访问，旋转延时也是不同的，因此旋转延时也取平均值，现在一般在 3ms 到 6ms 之间。平均旋转延时和磁盘转速有关，它用磁盘旋转一周所需时间的一半来表示。转速为 7200r/min 的磁盘其平均旋转时间为 4.17ms。

传送时间：磁盘的数据传送时间，除了与所传送的数据大小有关外，主要取决于磁盘的数据传输率。磁盘存储器在单位时间内向主机传送数据的字节数，称为**数据传输率**，数据传输率与存储设备和主机接口逻辑有关。从主机接口逻辑考虑，应有足够快的传送速度向设备接收/发送信息。从存储设备考虑，磁盘转速越快，数据传输率也就越高。假设磁盘旋转速度为每秒 r 转，每条磁道容量为 N 个字节，则数据传输率为

$$D = rN(\text{B/s})$$

若要传送的数据为 b 字节，则传送时间为

$$T = \frac{b}{rN}$$

于是，磁盘存储器总的平均访问时间 T_a 可表示成

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

其中 T_s 为平均寻道时间。

表 5-5 给出了当今典型的磁盘存储器的性能参数。

表 5-5 典型的磁盘存储器性能参数

特 性	Seagate Barracuda 180	Seagate CheetahX15-36LP	Toshiba DT01ACA05050	Toshiba HDD1242	Hitachi Microdrive
应用	大容量服务器	高性能服务器	桌面系统	便携式	手持设备
容量/GB	181.6	36.7	500	5	4
最小寻道时间/ms	0.8	0.3	0.6	—	1
平均寻道时间/ms	7.4	3.6	15.5	15	12
转速/(r/min)	7200	15 000	7200	4200	3600
平均旋转延时/ms	4.17	2	4.17	7.14	8.33
最大传输率/(MB/s)	160	522～709	140	66	7.2
每扇区字节数	512	512	4096	512	512
每道扇区数	793	485	—	63	—
盘面数	24	8	2	2	2
柱面数(每盘面磁道数)	24 247	18 851	—	10 350	—

【例 5.3】 设某磁盘组有 6 片磁盘，每片有两个记录面，最上最下两个面不用。存储区域内圈直径为 20mm，外圈直径为 80mm，道密度为 20 道/毫米，最内圈磁道位密度为 1 000 位/毫米，问：

(1) 盘组总存储容量是多少字节？

(2) 若该磁盘经格式化后，每个磁道有 12 个扇区，每扇区的容量为 512B，则该磁盘格式化容量为多少字节？

解：

(1) 每个盘面的磁道数＝20×(80－20)/2＝600(道)

每道的容量＝3.14×20×1000＝62 800(位)

盘组总存储容量＝盘面数×磁道数×道容量＝10×600×62 800/8＝47(GB)

(2) 磁盘格式化容量＝盘面数×磁道数×每道扇区数×每扇区容量

＝10×600×12×512＝36.8(GB)

【例 5.4】 设一个转速为 15 000 转/分(RPM)的磁盘，其平均寻道时间为 4ms，每磁道 500 扇区，每扇区 512B。现欲从该磁盘上读取一个由 2500 个扇区组成的总长为 1.28MB 的文件。

(1) 假设该文件随机分布在磁盘的不同扇区中，试计算读取该文件所需的总访问时间。

(2) 若该文件所占扇区分布在同一个柱面的 5 个磁道上，试计算读取该文件所需的总访问时间。

解：

（1）根据磁盘转速可以计算得到平均旋转延时为 2ms，对每个扇区而言，

访问时间＝平均寻道时间＋平均旋转延时＋传送时间

＝4ms＋2ms＋4/500ms

＝6.008ms

则读取该文件所需的总访问时间计算如下：

总访问时间＝2500×6.008ms＝15.02s

（2）由于 2500 个扇区正好占用了 5 个磁道，且这 5 个磁道又位于同一柱面上，因此寻道只需一次，时间为 4ms，因此读取该文件所需的总访问时间计算如下：

总访问时间 ＝4ms＋5×读取一个磁道的时间

＝4ms＋5×(2ms＋4ms)

＝0.034s

知识拓展

硬 盘 接 口

主机与硬盘存储器的连接是通过硬盘接口进行的。计算机中的硬盘接口主要包括 IDE 接口、SCSI、光纤通道和 USB 接口等。

现代 PC 的主板上主要集成的是 IDE 接口(见图 5-32)。IDE 是 Integrated Device Electronics 的简称，又被称为 ATA(Advanced Technology Attachment)。这两个名词都有厂商在使用，指的是相同的东西。IDE 之后又推出了 EIDE，即扩展的 IDE(Enhanced IDE)，而这个接口标准同时又被称为 Fast ATA。所不同的是 Fast ATA 是专指硬盘接口，而 EIDE 还制定了连接光盘等产品的标准。之后又陆续推出的速度更快的接口，名称都留有 ATA 的字样，如 Ultra ATA 以及传输速度分别达到 66MB/s、100MB/s、133MB/s 的 ATA/66、ATA/100、ATA/133 等。

以上传统的 ATA 接口均采用并行传输方式，现在又出现了串行 ATA(Serial ATA，SATA)，如图 5-33 所示。其最大数据传输率更进一步提高到了 150MB/s，将来还会提高到 300MB/s。而且其接口非常小巧，排线也很细，有利于机箱内部空气流动从而加强散热效果，也使机箱内部显得不太凌乱。与并行 ATA 相比，SATA 还有一大优点就是支持硬盘的热插拔。

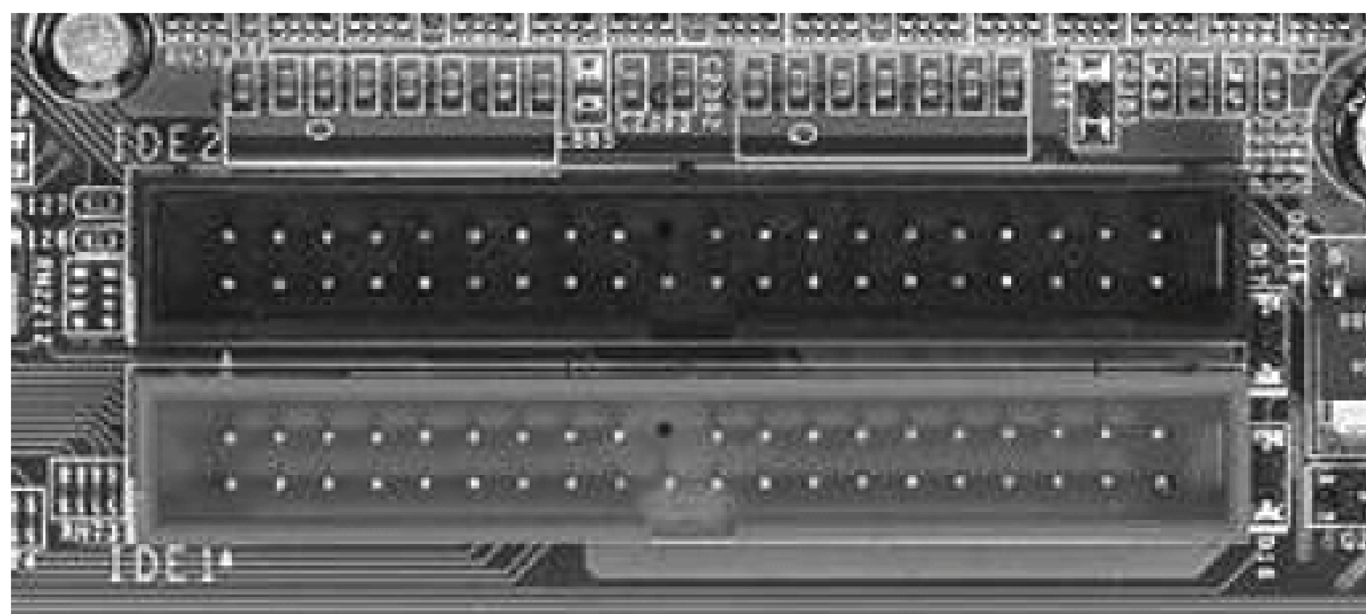


图 5-32 IDE 接口

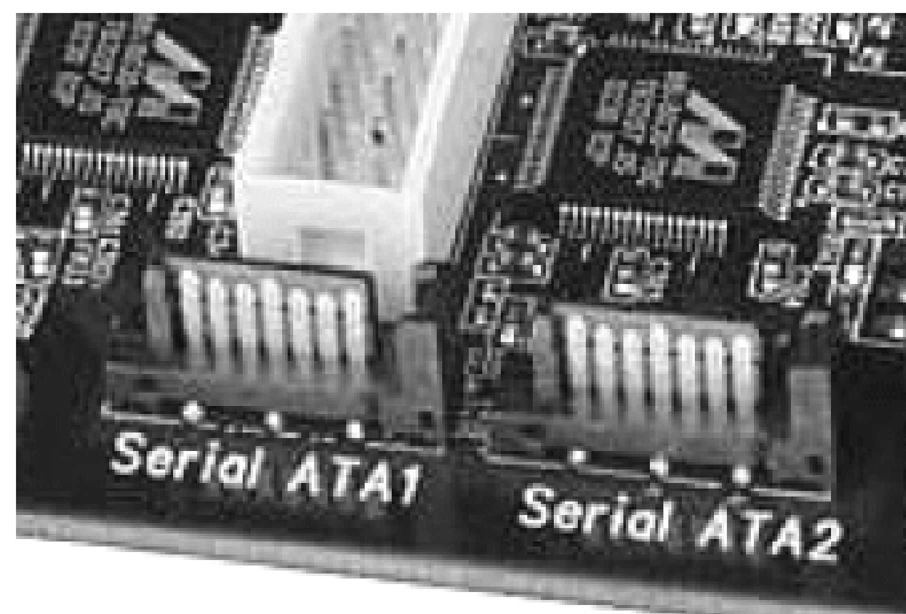


图 5-33 串行 ATA 接口

SCSI 的英文全称为 Small Computer System Interface(小型计算机系统接口)，是同 IDE(ATA)完全不同的接口。SCSI 并不是专门为硬盘设计的接口，而是一种广泛应用于小

型机上连接各种具有SCSI设备的高速数据传输接口。SCSI具有多任务、带宽大、CPU占用率低以及热插拔等优点。但SCSI硬盘较高的价格使得它很难在PC中普及,因此SCSI硬盘主要应用于中、高端服务器和高档工作站中。

SCSI从诞生到现在已经历了三十多年的发展,先后衍生出了SCSI-1、Fast SCSI、FAST-WIDE-SCSI-2、Ultra SCSI、Ultra2 SCSI、Ultra160 SCSI、Ultra320 SCSI等,现在市场中占据主流的是传输速度分别达到160MB/s、320MB/s的Ultra160 SCSI、Ultra320 SCSI接口产品。

光纤通道(Fibre Channel)和SCSI一样最初也不是为硬盘设计开发的接口技术,是专门为网络系统设计的,但随着存储系统对速度的需求,才逐渐应用到硬盘系统中。光纤通道硬盘是为提高多硬盘存储系统的速度和灵活性才开发的,它的出现大大提高了多硬盘系统的通信速度。光纤通道的主要特性有:热插拔性、高速带宽、远程连接、连接设备数量大等。

光纤通道是为在像服务器这样的多硬盘系统环境而设计的,能满足高端工作站、服务器、海量存储子网络、外设间通过集线器/交换机和点对点连接进行双向串行数据通信等系统对高数据传输率的要求。

USB接口不是专为硬盘设计的接口标准。目前在PC上,USB接口用途广泛,如可以用于连接移动硬盘。有关USB接口的内容在第6章详细讲述。

5.3.2　磁带存储器

磁带存储器也属于一种磁介质存储器,与磁盘存储器一样,它也是通过磁记录原理来存储信息的。但与磁盘不同的是,它是通过在一条柔韧的聚酯薄膜带上涂上一层磁性材料来记录信息的。磁带卷在两个可以来回转动的转轴上,并固定装入一个盒中,组成盒式磁带。计算机所使用的磁带和磁带机与家用的磁带录音机相似,也是通过磁带在一个固定磁头下的平行移动来完成数据存取的。计算机中所用的磁带宽度从0.15in(0.38cm)到0.5in(1.27cm),长度从600in(182m)到2400in(728m)不等。

磁带上的磁道是沿磁带运动方向平行排列的。早期磁带系统一般使用9个磁道,每次存取一个字节,其中8个磁道构成一个有效字节信息,第9磁道上是附加的奇偶校验位。后来的磁带系统使用18或36个磁道,这对应于数字的一个字或双字。这种记录格式称为并行记录。然而,现代大多数磁带系统使用串行记录方式,数据作为一系列的二进制位串沿同一磁道顺序存储,就如同磁盘在同一磁道上顺序存储数据一样。随着技术的发展,磁带上的磁道数可以达到几百道,这大大提高了磁带存储器的容量。

串行记录磁带使用一种被称为蛇形记录(Serpentine Recording)的记录方式。按此方式,数据从一个磁带的头部开始,沿一个磁道从头到尾记录,到达磁带尾部时,再沿另一磁道从尾到头记录,再次回到磁带头部时,又沿第三个磁道从头到尾记录,如此往复,如图5-34所示。

磁带在数据组织上也是采用分块的方式。图5-35(a)和图5-35(b)分别是一种采用并行记录的0.5in 9道启停式磁带和采用串行记录的0.25in数据流磁带的数据记录格式。

0.5in 9道启停式磁带是一种国际上通用的标准磁带。每盘带均设有卷头标BOT和卷尾标EOT,标记用一块矩形金属反光薄膜制成,光电检测元件可检测到这两个标记,表示记录的开始和结束。磁带上留有间隙G(3.75in)或g(0.6in),后者为数据块间的间隙,它们取

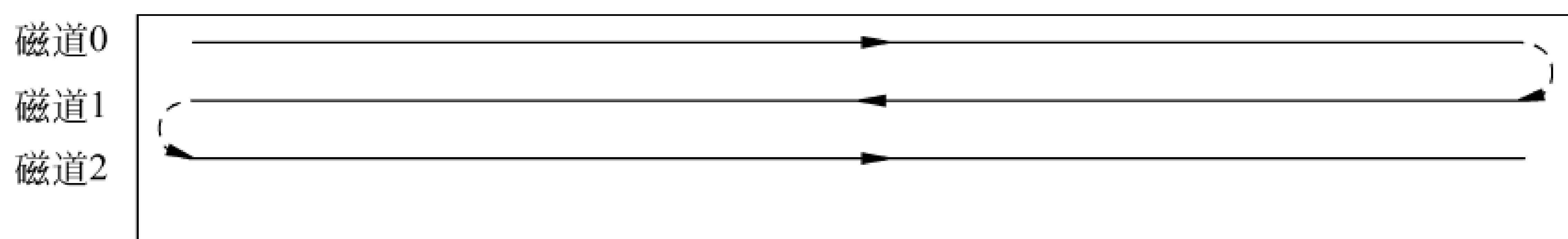
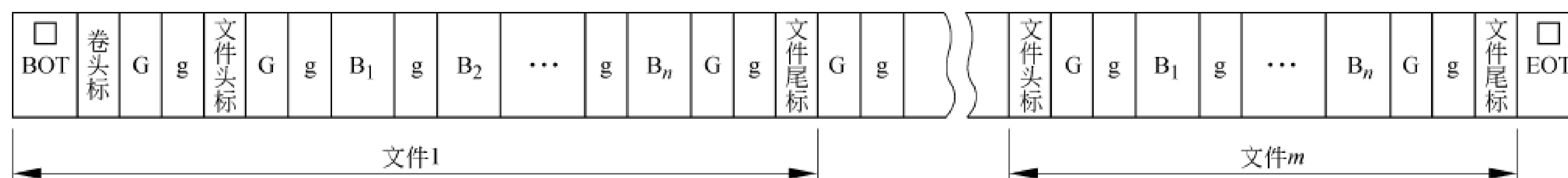
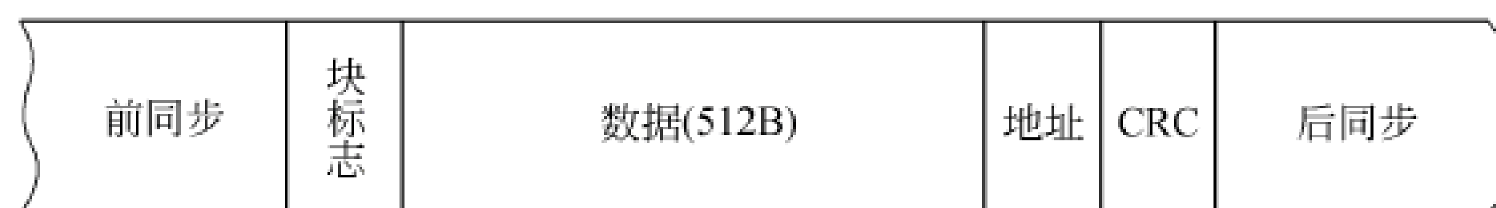


图 5-34 磁带的蛇形记录方式



(a) 0.5in 启停式磁带数据记录格式



(b) 0.25in 数据流磁带数据记录格式

图 5-35 磁带记录格式

决于磁带机的快启停性能。

信息可用两种形式存储，一种是文件形式，一盘带可记录若干个文件，一个文件又分若干数据块(B*i*)，每个文件始末有文件头标和文件尾标，卷头标、索引、文件头标、文件尾标均为 80B，其内容视操作系统而定。第二种是数据块形式，磁带可在数据块之间启停，进行数据传输。在 9 道带中，8 位是数据磁道，存储一个字节，另一位是这一字节的奇偶校验位，叫做横向奇偶校验码。在每一数据块 B*i* 内部，沿走带方向每条磁道还有 CRC 校验码。

0.25in 盒式数据流磁带也是一种通用的标准磁带。其中 9 道磁带记录格式包括前同步、数据块标志(1B)、用户数据(512B)、地址号(4B)、CRC 校验码(2B)和后同步。

为提高磁带存储器的访问速度，磁头能同时对几个相邻磁道(通常是 2～8 个磁道)进行读写操作。数据仍是沿各磁道蛇形串行记录，但数据块不是在同一磁道上顺序存放的，而是在相邻磁道依序排列的。如图 5-36 所示是一个能同时进行 4 个磁道读写的磁带的块分布图。

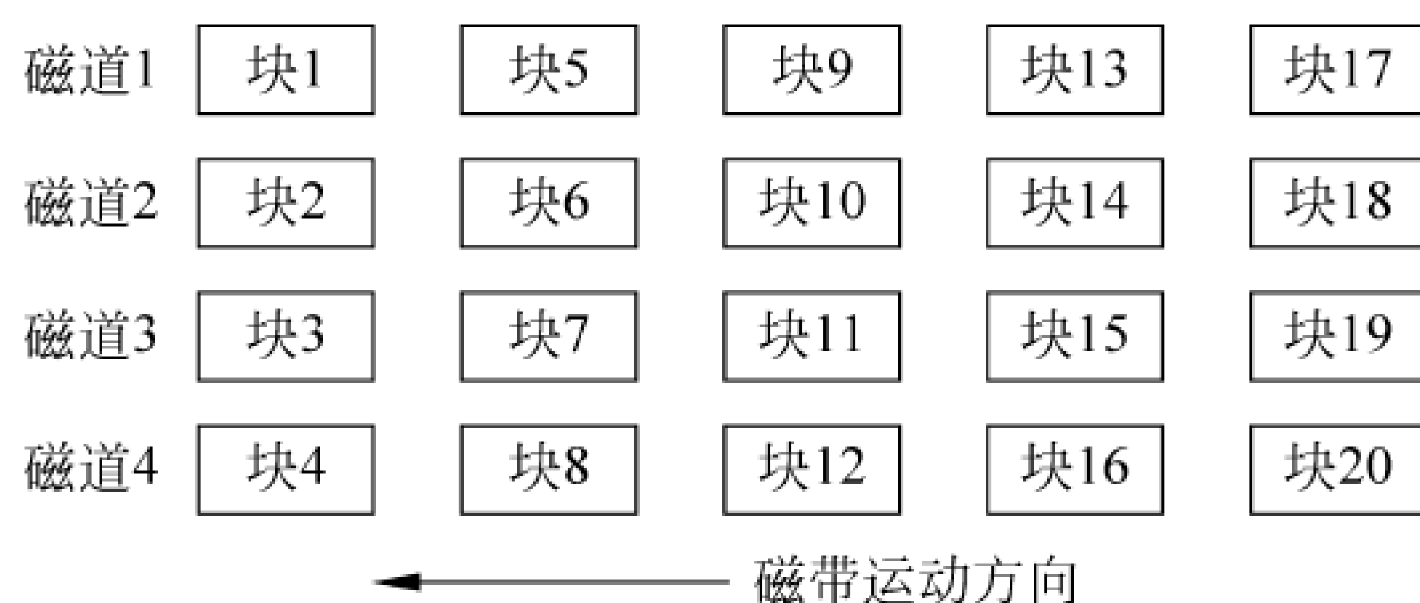


图 5-36 磁带的块分布

表 5-6 给出了几个磁带的系统参数。

表 5-6 几个磁带的系统参数

系统参数	DLT4000	DLT8000	SDLT600
容量/GB	20	40	300
数据速率/(MB/s)	1.5	6	36
位密度/(Kb/cm)	32.3	38.6	92
磁道密度/(道/cm)	101	164	587
磁带长度/m	549	549	597
磁带宽度/cm	1.27	1.27	1.27
磁道数	128	208	448
同时读写的磁道数	2	4	8

磁带驱动器是一种顺序存取设备。若磁头当前定位于块1,而需读块N,则它必须依次读物理块$1\sim N-1$,每次一个块。若磁头当前定位已超越所需访问的块时,它必须倒带,从前面开始重新向前读。

磁带存储器是一种低价格、慢速、大容量的辅助存储器,在计算机系统中常用于数据备份。

5.3.3 光盘存储器

光盘存储器是一种采用光存储技术存储信息的存储器,它采用聚焦激光束在盘式介质上非接触地记录高密度信息,以介质材料的光学性质(如反射率、偏振方向)的变化来表示所存储信息的"1"或"0"。由于光盘存储器容量大、价格低、携带方便及交换性好等特点,已成为计算机中一种重要的辅助存储器,也是现代多媒体计算机(MPC)不可或缺的存储设备。

光存储技术源于20世纪70年代。1972年,Philips公司设计出世界上第一个能播放模拟电视信号的光盘系统。1978年,世界上第一台商品化的激光视盘机(Laser Vision,LV)由Philips公司推出,其原理是仿效声音唱片的形式,把图像和伴音信号记录在圆盘上,用激光束检测盘上记录的信息,将其转换成电信号,经处理后还原成视频和音频信号,由电视机显示图像和发出声音。1981年,Philips公司和Sony公司携手推出了数字激光唱盘(Compact Disc-Digital Audio,CD-DA),并为此制定了光盘技术领域非常重要的基础性技术文件——《红皮书标准》。

1985年,Philips和Sony的研究人员在经过几年的努力后终于解决了光盘上只能记录数字音乐信息,而不能记录计算机文件信息的问题。具体来说就是解决如何在光盘上划分地址,以便计算机系统可以根据地址编号随时存取数据和降低光盘数据存取误码率问题。为此他们公布了在光盘上记录计算数据的《黄皮书标准》。后来国际标准化组织(ISO)又对该标准进行了完善,发布了ISO 9660标准。这样,CD-ROM便进入了计算机,并很快得到了广泛的应用,CD-ROM已成为现代多媒体计算机中的标准配置之一。随后,研究人员们一方面努力提高CD-ROM的读取速度,由最初的2倍速、4倍速(MPC3标准)发展到今天的52倍速;另一方面又进一步推出了用于计算机中可读写的光盘和DVD等,巩固和确立了光盘存储器在计算机辅助存储器中的重要地位。

1. 光盘存储器的分类

按光盘可擦写性分类主要包括只读型光盘和可擦写型光盘。

只读型光盘所存储的信息由光盘制造厂家预先用模板一次性将信息写入，以后只能读出数据而不能再写入任何数据。按照盘片内容所采用的数据格式的不同，又可以将盘片分为CD-DA、CD-I、Video-CD、CD-ROM、DVD等。

可擦写型光盘是由制造厂家提供空盘片，用户可以使用刻录光驱将自己的数据刻写到光盘上，它包括CD-R、CD-RW和相变光盘及磁光盘等。

常见的光盘种类、功能及相关标准如表5-7所示。

表5-7 常见光盘种类、功能及相关标准

光盘种类	数据容量	执行标准	出现时间	功能说明
CD-DA	最大播放音乐时长74min	红皮书	1982年	CD系列光盘的始祖，由Philips和Sony于1982年正式发布，主要用于音乐存储
CD-ROM	存储650MB计算机数据	黄皮书	1985年	由Philips和Sony联合制定，定义了存储计算机数据的规范，规定了地址数据结构、数据纠错、扇区大小等，使光存储进入计算机领域
CD-I	760MB	绿皮书	1986年	Philips和Sony针对消费电子市场推出的一种交互式多媒体数据存储格式，使之能同步播放声音、影像及其他信息如文字等
CD-R	700MB	橙皮书	1992年	一次刻录型的光盘片，不管数据是否填满盘片，只能写入一次，即使还剩余空间，也不能再写
CD-RW	700MB	橙皮书的第三部分	1996年	刻录方式与CD-R相同，区别是其可以擦除和重复写入，CD-RW驱动器完全兼容CD-R盘片
Video CD	70min MPEG1格式数据	白皮书	1993年	可存储按MPEG1格式压缩的视频和音频信息，主要应用播放电影等
DVD	存储高达17GB数据	ISO/IEC 16448	1996年	全称是数字视盘(Digital Video Disk)，将计算机和家庭娱乐融合起来，DVD驱动器可以识别各种CD盘片，已有取代CD-ROM之势

2. CD-ROM

标准CD-ROM盘片的直径为120mm，中心装卡孔径为15mm，厚度为1.2mm，重量为14～18g，其基质由树脂(如聚碳酸酯)制成，数据信息以一系列微凹坑的样式刻录在光盘表面上。CD-ROM光盘在制作时，首先用精密聚焦的高强度激光束制造一个母盘，然后以母盘作为模板压印出聚碳酸酯的复制品，再在凹坑表面上镀一层高反射材料(铝或金)，最后在外层上涂一层丙烯酸树脂以防灰尘或划伤，如图5-37所示。

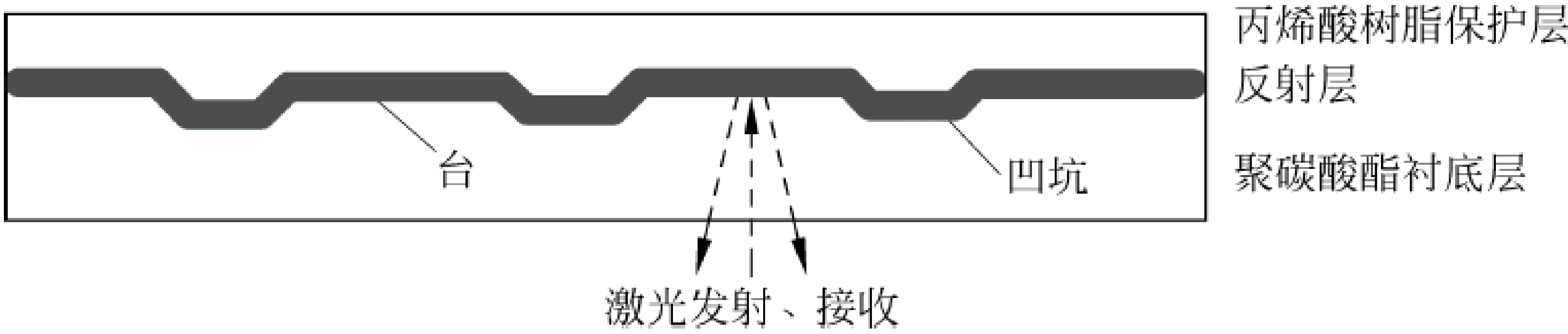


图5-37 CD-ROM的组成结构

CD-ROM是通过安装在光盘驱动器内的激光头来读取盘片上的信息的。当盘片转动

并经过激光头时,激光头能产生可以穿过透明的聚碳酸酯层的低强度激光束。激光束照射到盘片的不同区域时,反射的激光强度发生变化。具体来说,当激光束照射在凹坑上时,由于凹坑表面有些不平,光被散射,反射回的光强度变低。凹坑之间的区域称为台(land),台的表面光滑平坦,反射回的光强度高。光传感器将检测到的这种光强变化转换成数字信号。传感器以固定的间隔检测盘表面,一个凹坑的开始或结束表示存储了一位二进制“1”;间隔之间无标高变动出现时,记录的是“0”。

CD-ROM 与磁盘在数据记录方式上有所不同。磁盘是由一个个同心圆的磁道组成的。而 CD-ROM 却不同,它在整个盘面上只有一条螺旋式轨道,由靠近中心处开始,逐圈向外旋转直到盘的外沿。靠外的扇区与靠内的扇区具有相同的长度,于是,按同样大小的段分组的信息可以均匀分布在整个盘上。

CD-ROM 的数据存储也是以块为单位进行组织的。典型的块格式如图 5-38 所示。它由下列字段组成。

图 5-38 CD-ROM 数据块格式

(1) Sync:同步字段,标志一个块的开始。由 12 个字节组成,第 1 个字节为全 0,第 2~第 11 个字节为全 1,第 12 个字节为全 0。

(2) ID:标识字段,包含块地址和模式字节。模式 0 表示一个空的数据域,模式 1 表示使用纠错码和 2048B 的数据,模式 2 表示不带纠错码的 2336B 的用户数据。

(3) Data:用户数据域。

(4) Auxiliary:此辅助域在模式 2 下是附加用户数据,在模式 1 下是 288B 的纠错码。

与传统的硬盘相比,CD-ROM 有两个主要优点:

(1) 价格便宜,单片容量 650MB,单位价格比硬盘低得多。

(2) 光盘是可交换的,可以用于批量制作软件产品,也方便用户存档和交换等。

CD-ROM 的缺点是:

(1) 它是只读的,不能修改。

(2) 存取时间比磁盘驱动器长得多。

CD-ROM 是通过专门的 CD-ROM 驱动器(即通常所说的光驱)来进行读操作的。CD-ROM 驱动器一方面完成对光盘的读操作,另一方面与主机相接口。常见的 CD-ROM 驱动器接口标准主要有三种:

1) 专用接口

由各 CD-ROM 驱动器生产厂家提供的专用接口卡将 CD-ROM 与主机连接起来。目前专用接口正逐步被取代。

2) IDE 接口

IDE(EIDE)接口的 CD-ROM 驱动器直接插在计算机主板上的 IDE 或 EIDE 插口上,无须配置总线接口卡,这也是目前微型计算机中普遍采用的一种接口方式。

3）SCSI

小型计算机系统接口(Small Computer System Interface,SCSI)是目前比较流行的输入输出接口标准,相对来说,SCSI 比 IDE 接口速度更快。

3. CD-R

CD-R(Compact Disk Recordable)是一种一次写、多次读的可刻录光盘系统,它由 CD-R 盘片和刻录光驱组成。

CD-R 光盘与普通 CD-ROM 光盘在外观尺寸、记载数据的方式等方面是相同的,也同样是利用激光束的反射原理来读取信息的。但与 CD-ROM 不同的是,在 CD-R 光盘表面除了含有聚碳酸酯层、反射层和丙烯酸树脂保护层外,另外还在聚碳酸酯层和反射层之间加上了一个有机染料记录层。

当使用 CD-R 刻录光驱对空白盘片进行刻录时,是将写激光束照射到有机染料记录层上,激光照射时产生的热量将有机染料烧熔,并使其产生光痕。光痕会使今后读激光束改变光的反射率,从而达到一次刻录改写信息的目的。

4. CD-RW

CD-RW(Compact Disk ReWritable)光存储系统是在 CD-R 基础上进一步发展起来的,是一种多次写、多次读的可重复擦写的光存储系统。

CD-RW 光盘结构与 CD-ROM 基本相同,只是在盘片中增加了可改写的染色层。读写数据采用相变(Phase Change)技术。相变技术利用物质的状态变化进行数据的读、写和擦除。CD-RW 盘片内部镀上一层一定厚度的薄膜即相变记录层。相变记录层由一种银合金材料组成,随加热温度的不同,它可以形成晶体,也可以形成非晶体。因此,适当调整加热温度就可以自由地控制记录层的结晶状态。在晶体状态中原子整齐排列,光反射率高;相反,在非晶体状态中原子排列不整齐,光反射率低。对 CD-RW 的读、写和擦除正是利用光反射率的这种变化来实现的。由于材料的因素,晶体状态改变的次数有限,也使得 CD-RW 盘片的擦写次数有限。

CD-RW 盘片中的相变记录层的记录膜在出厂时处于晶体状态,写入时用强的激光束照射使之变为非晶体状态,如果此时中止激光照射,记录膜温度急剧下降,写入数据的区域便稳定在非结晶状态,数据被写入。读出时用弱的激光束照射记录区,并根据反射光的反射率判别是 0 还是 1。仅用弱光照射时,记录膜记录的数据不会被破坏,这与普通光驱读取光盘的原理是一样的。在擦除数据时,用中等强度的激光束照射记录膜,使其温度上升少许,记录膜又返回晶体状态,数据被擦除。

对 CD-RW 盘片的读写操作是通过 CD-RW 刻录机完成的。目前的 CD-RW 刻录机兼容 CD-ROM 和 CD-R 盘片,它分为内置式和外置式两种。在与主机接口上,内置式刻录机主要通过 IDE、SCSI 等接口连接,而外置式刻录机通过计算机的外部并行接口连接。

5. DVD

DVD 的英文全名是 Digital Video Disk,即数字视频光盘。DVD 不仅仅用来存储视频数据,还可以用来存储其他类型的数据,因此 DVD 又为 Digital Versatile Disk,即数字通用

盘，是一种能够保存视频、音频和计算机数据的容量更大、运行速度更快的采用了 MPEG2 压缩标准的光盘。

DVD 采用了类似 CD-ROM 的技术，但是可以提供更高的存储容量。从表面上看，DVD 盘片与 CD-ROM 盘片很相似，其直径为 80mm 或 120mm，厚度为 1.2mm。但实质上，两者之间有本质的差别。相对于 CD-ROM 光盘 650MB 的存储容量，DVD 光盘的存储容量可以高达 17GB。另外在读盘速度方面，CD-ROM 的单倍速传输速度是 150KB/s，而 DVD 的单倍速传输速度是 1358KB/s。

图 5-39 是 DVD 和 CD-ROM 盘片数据记录道和凹坑情况的比较。从图 5-39 中可以看出，CD-ROM 盘的道间距为 1.6μm，而 DVD 盘的道间距为 0.74μm；CD-ROM 盘的最小凹坑为 0.83μm，而 DVD 盘的最小凹坑为 0.4μm。DVD 盘片的道密度和凹坑密度都远高于 CD 盘片。单从这两方面的改进，就使 DVD 的单片单层容量提高到 CD-ROM 的 7 倍多，可达 4.7GB。

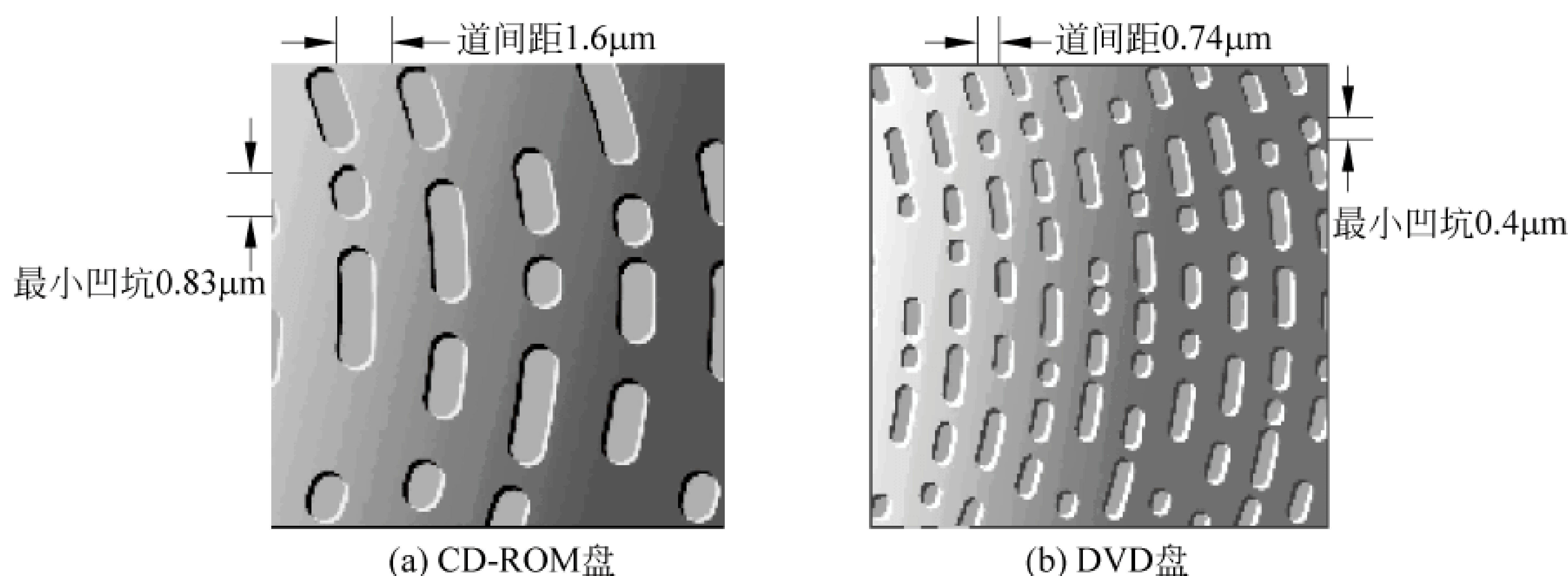


图 5-39　DVD 盘与 CD 盘的凹坑密度比较

DVD 盘片分为单面单层、单面双层、双面单层和双面双层 4 种物理结构。因此，可以将 DVD 盘片分为 4 种规格，分别是 DVD-5、DVD-9、DVD-10 和 DVD-18，它们的容量分别如表 5-8 所示。

表 5-8　4 种 DVD 盘片比较

盘片类型	盘片直径/cm	面数/层数	容量/GB
DVD-5	12	单面单层	4.7
DVD-9	12	单面双层	8.5
DVD-10	12	双面单层	9.4
DVD-18	12	双面双层	17

无论是单层盘还是双层盘，都是由两片基底组成，每片基底的厚度均为 0.6mm，因此 DVD 盘片的厚度为 1.2mm。对于单面盘来说，只有下层基底包含数据，上层基底没有数据。而双面盘的上下两层基底均包含数据。

从读写功能上，DVD 盘片也分为两类：只读型和可擦写型。只读型 DVD 包括用于计算机辅助存储器的 DVD-ROM 和用于存储音频、视频的 DVD-Audio 及 DVD-Video；可擦写型 DVD 主要有一次可刻录的 DVD-R 和多次可擦除、可改写的 DVD-RW 等。

5.4 RAID 技术

计算机技术发展到今天，使得人们可以在包括微型计算机在内的各种机器中配置大容量磁盘。而在此之前，只有在一些大型计算机系统中根据其应用的需要才配备大容量磁盘系统。那时，大容量磁盘的价格还相当高，对工作环境的要求也非常高。虽然那时磁盘的可靠性已很高，但对磁盘不恰当的使用仍然容易造成磁盘的损坏。一旦磁盘损坏，就会造成不可修复的磁盘错误，将使其在商业、科研及学术等领域造成不可估量的损失。因此使用单一的磁盘存储重要的数据总存在安全问题。解决数据存储安全问题的主要方法是使用磁带机等设备进行数据备份。这种方法虽然可以在一定程度上提高数据的安全性，但一方面数据的备份需要科学合理的安排和严格的制度来保障；另一方面今后对数据的检索工作相当烦琐。

1988 年，美国加州大学 Berkeley 分校的 David Patterson、Garth Gibson 和 Randy Katz 三人发表了一篇题为 *A Case of Redundant Array of Inexpensive Disks*(《廉价磁盘冗余阵列方案》)的论文，首次提出了 RAID 一词。在论文中他们提出将多个小容量、价格低廉的磁盘进行有机组合，来替代通常在大型计算机中使用的昂贵的大容量磁盘系统，并使其具有更好的性能和更高的可靠性。在这篇论文中，Patterson、Gibson 和 Katz 还定义了 5 种类型(称为级，level)的 RAID，每一级 RAID 都具有不同的性能和可靠性。以前这些级的编号是从 1 到 5，后来人们又定义了 RAID 的第 0 级和第 6 级。所以 RAID 主要分为 7 个级，即从第 0 级到第 6 级。当然，一些研究机构和公司还定义了其他一些 RAID 级，但 7 级 RAID 是业界普遍认同的标准。

总的来说，RAID 的设计思想是通过在多个硬盘上(又称为磁盘阵列)同时存取数据来大幅提高磁盘存储系统的数据吞吐量的，而且在一些 RAID 模式中通过较为完备的相互校验/恢复的措施，甚至是直接相互的镜像备份，以提高 RAID 系统的容错度，从而提高了磁盘存储系统的安全性和可靠性。RAID 的这一设计思想很快被接受，从此 RAID 技术得到了广泛应用，数据存储进入了更快速、更安全、更廉价的新时代。

表 5-9 对 RAID 分级进行了概括。

表 5-9 RAID 分级

RAID 级	描　　述	磁盘数	容错性能	并行 I/O 响应
0	无容错的分块磁盘阵列	N	无容错	有
1	磁盘镜像	$2N$	容错性最好	有
2	专用海明校验盘位分布磁盘系统	$N+m$	允许一个磁盘失效	无
3	专用奇偶校验盘位分布磁盘系统	$N+1$	允许一个磁盘失效	无
4	专用奇偶校验盘分块独立存取磁盘系统	$N+1$	允许一个磁盘失效	有
5	分散校验分块独立存取磁盘系统	N	允许一个磁盘失效	有
6	分散双校验分块独立存取磁盘系统	N	允许两个磁盘失效	有

1. RAID0

RAID0 全称是 Striped disk array without fault tolerance(没有容错的条带磁盘阵列)，

其构架如图 5-40 所示。

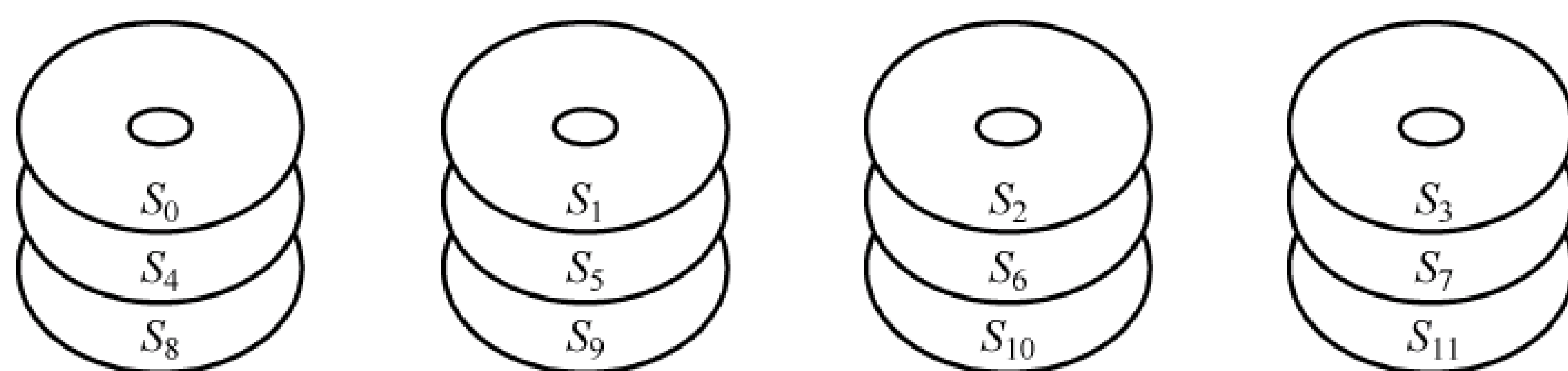


图 5-40　RAID0 的分条带数据组织

图 5-40 中一个圆盘就是一个磁盘(以下同),磁盘以条带(Strip)的形式划分,每个条带是一些物理的块、扇区或其他单位。所有的磁盘组成一个逻辑磁盘,系统数据和用户数据被看成存储在这个逻辑磁盘上。逻辑磁盘上的一个个条带数据以轮转方式映射到连续的阵列磁盘中。例如,在一个由 N 个磁盘组成的阵列中,逻辑磁盘的第 1～第 N 个条带数据按顺序依次分布在第 1～第 N 个磁盘的第 1 个条带上,第 $N+1$～第 $2N$ 个条带数据按顺序依次分布在第 1～第 N 个磁盘的第 2 个条带上,…,以此类推。这种布局的优点是,如果单个 I/O 请求由多个逻辑相邻的条带组成,则对多达 N 个条带的请求可以并行处理,这样可以大大提高 I/O 的数据传输率。

实际上,RAID0 有些类似在计算机主存中采用的交叉存储技术,通过使用多个磁盘的并行存取提高主机对磁盘的访问速度。每个磁盘都有自己独立的访问控制电路,能够独立地进行数据的传输。而逻辑磁盘和物理磁盘间的数据映射则在一个磁盘阵列管理系统(由软件或硬件实现)的统一控制下完成。

不过,RAID0 不是 RAID 家族中的真正成员,因为它没有数据冗余能力。由于没有采用备份或校验恢复技术,在 RAID0 阵列中任何一个磁盘损坏就会导致整个磁盘阵列数据的损坏,因为数据都是分布存储的。而且,RAID0 磁盘阵列的整体可靠性会随着磁盘数量的增加而降低。例如,如果阵列由 5 个磁盘组成,每个磁盘的设计寿命为 50 000 小时,那么整个系统的期望设计寿命为 50 000/5=10 000 小时。当磁盘数量进一步增加时,磁盘阵列失效的概率会随之增加,最后达到某个不确定的值。

2. RAID1

RAID1 又称为磁盘镜像(Disk Mirroring),是所有的 RAID 级中具有最佳失效保护的一种方案。它使用两组互为镜像的磁盘进行简单的完全数据备份,从而实现数据冗余,如图 5-41 所示。

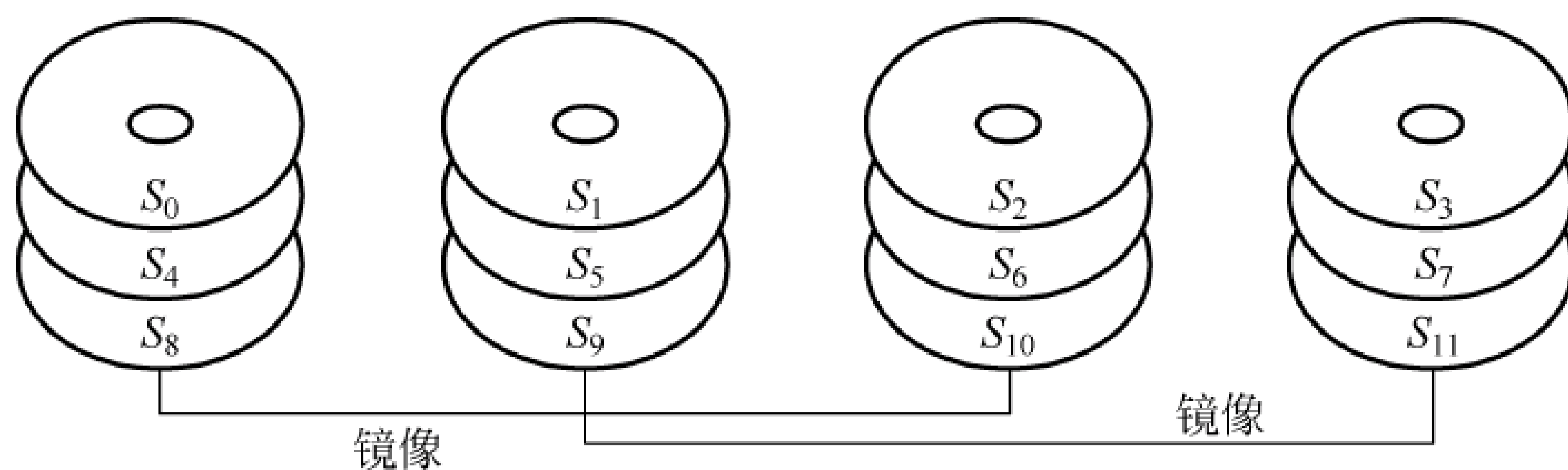


图 5-41　RAID1 的磁盘镜像

图 5-41 中包含了两组相同的磁盘阵列。RAID1 在每次写入时，都会同时将数据写入两组磁盘中，使两组磁盘的数据保持完全相同，以实现磁盘阵列的高可靠性。

RAID1 也采用与 RAID0 相同的条带数据划分，即在每组内，所有磁盘以条带方式进行数据组织，以保持与 RAID0 同样的高性能。

与 RAID0 相比，RAID1 具有以下特点：

(1) 一个读请求可由包含请求数据的两组磁盘中的某一个提供服务，只要它的寻道时间加旋转延迟较小。因此，RAID1 具有比 RAID0 更优的读性能。尤其是在一些面向事务的应用场合，当需要对磁盘进行大量数据读取时，RAID1 的性能甚至能接近达到 RAID0 性能的两倍。

(2) 一个写请求需要更新两组磁盘中两个对应的条带，而这组磁盘的写操作可以并行进行。因此，写性能由两者中较慢的一个写操作来决定，即包含较大的寻道时间和旋转延迟的那一个写。

(3) 恢复一个损坏的磁盘很简单。当一个磁盘损坏时，数据仍能从与之镜像的磁盘中读取。

RAID1 的主要缺点是价格昂贵，它需要支持两倍于逻辑磁盘的磁盘空间。因此，RAID1 的配置常用在存储关键的系统数据和用户数据的场合中。在这种情况下，RAID1 对所有的数据提供实时备份，即使一个磁盘损坏，所有的关键数据仍能立即可用。

3. RAID2

RAID1 虽然同时具有高可靠性和高性能，但它的主要问题是成本太大，需要整整两倍于实际所需的磁盘数量才能达到数据冗余。更好的方式是只使用磁盘组中的一个或几个磁盘用于数据冗余或数据校验之用。RAID2 就定义了这些方法中的一种。

RAID2 称为海明码校验(Hamming Code ECC)，它将磁盘进行条带划分的方法运用了极端的情形。它在每个条带中只写入一位二进制位，而不是采用像 RAID0 和 RAID1 中的数据块。这样的话，如果以字节为单位进行数据组织，则一个磁盘阵列中至少需要 8 个磁盘用于存储数据信息。RAID2 采用了海明纠错码进行数据校验。与数据磁盘相对应，磁盘阵列中还需要一组磁盘用于存储纠错码信息，如图 5-42 所示。

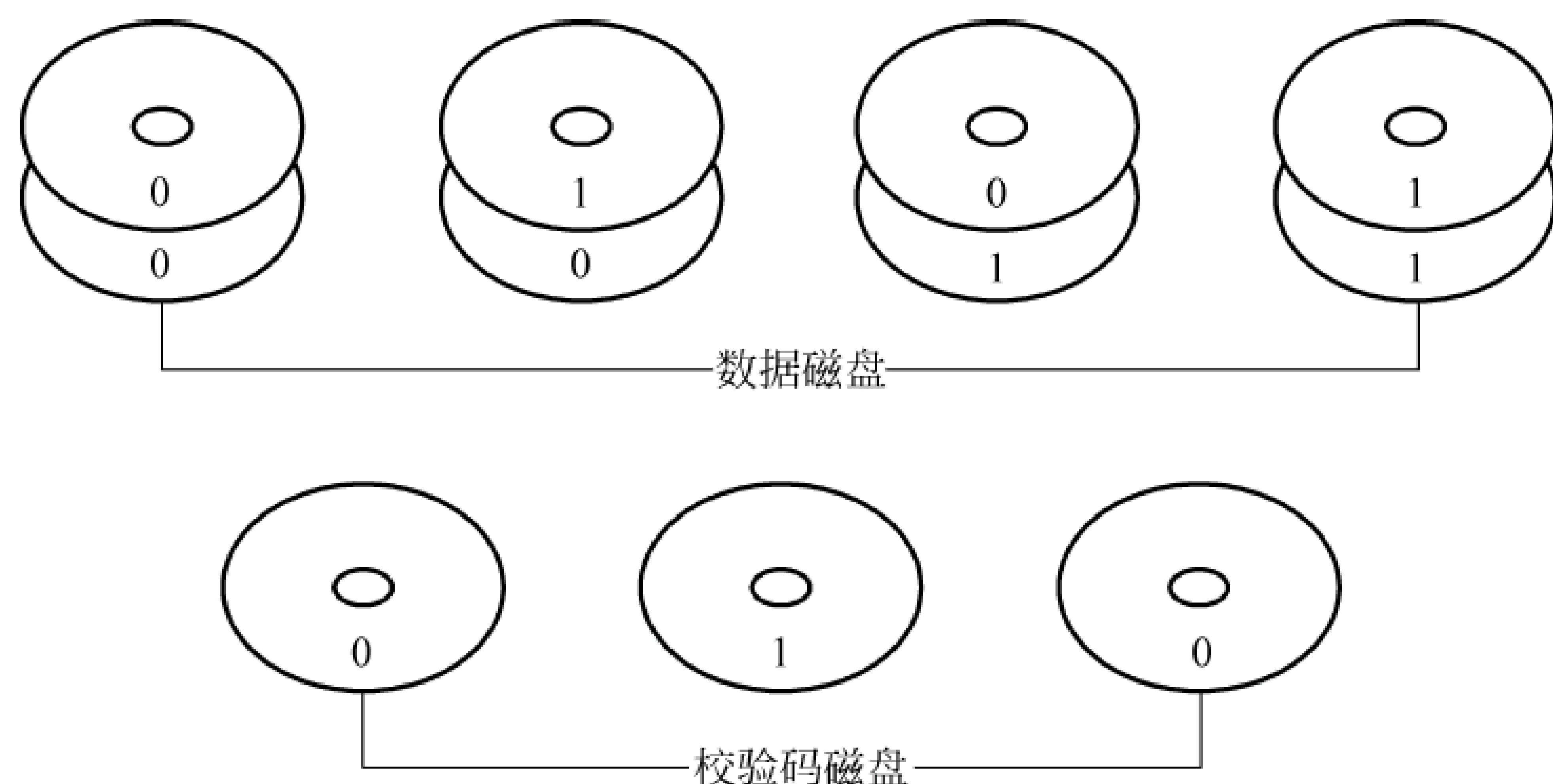


图 5-42　RAID2 的采用海明码校验的分条带数据组织

纠错码所需的磁盘数量取决于所采用的海明纠错码所需的校验位数。无论是数据盘还是校验盘，只要有一个磁盘损坏，其中的数据都可以通过海明码来重建恢复。

因为每个磁盘都只写入 1 位数据，因此整个 RAID2 的磁盘组就好像是一个大型的数据磁盘。所有的磁盘数据（包括数据盘和校验盘）都必须严格地同步，否则数据变乱会使得海明码起不到校验的作用。由于系统对磁盘的读写是按位并行执行的，因此数据传输率高。但由于生成海明码较为耗时，所以 RAID2 对大多数商业应用来说速度太慢。事实上，今天大多数磁盘驱动器都有内置的 CRC 纠错功能。对于单个磁盘和驱动器具有高可靠性的情况下，RAID2 就没有太大的应用意义了。

4. RAID3

RAID3 称为带校验的并行传输（Parallel Transfer with Parity）。像 RAID2 一样，RAID3 是按照每次一位的方式将数据交错分配到各个数据盘的条带上。但是，与 RAID2 所不同的是，RAID3 只使用一个磁盘来存储一个简单的奇偶校验位，如图 5-43 所示。这种奇偶校验位可以使用简单的硬件快速计算出来，具体方法是对分布在每一个磁盘上的每一个数据位（D_i）进行异或运算（对偶校验），生成的偶校验位存储到校验盘上。例如，假设磁盘组共有 8 个数据磁盘，分布在这 8 个盘上的数据位分别为 $D_0 \sim D_7$，则偶检验位为

$$P = D_0 \oplus D_1 \oplus D_2 \oplus D_3 \oplus D_4 \oplus D_5 \oplus D_6 \oplus D_7$$

其中 P 位存储到校验盘中。

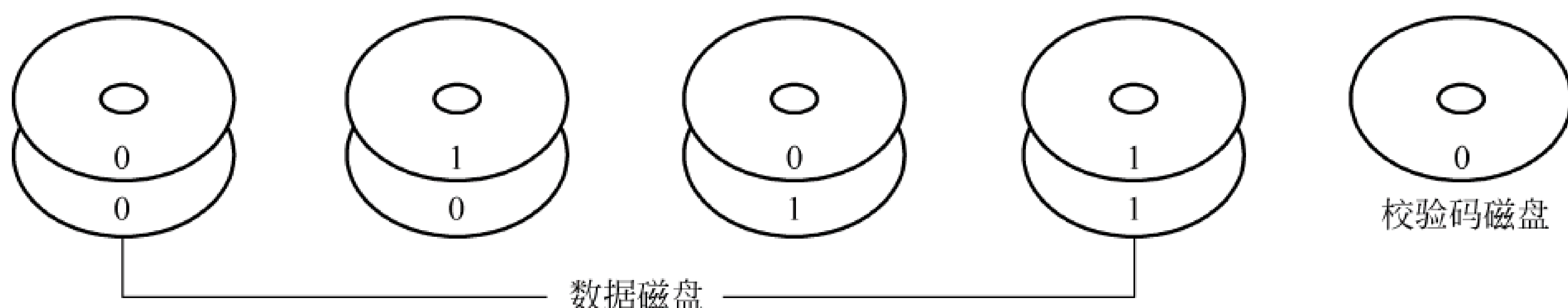


图 5-43　RAID3 的采用奇偶校验码的分条带数据组织

利用相同的方法可以在确认了某一磁盘损坏的情况下，很方便地对该磁盘数据进行重建。例如，假设上例中的 5 号磁盘损坏了，并且被替换掉。用户可以通过对其他剩余的数据磁盘和校验磁盘的对应数据进行以下偶校验运算，即可重新生成损坏了的磁盘上的数据：

$$D_5 = D_0 \oplus D_1 \oplus D_2 \oplus D_3 \oplus D_4 \oplus D_6 \oplus D_7 \oplus P$$

当然，RAID3 也要求使用与 RAID2 同样的数据复制方法和同步操作，以免造成数据的混乱。由于只使用了一个校验盘用于数据的恢复，因此，RAID3 比 RAID1 和 RAID2 都更经济。

5. RAID4

RAID2 和 RAID3 都采用了将数据以位为单位分布到各个数据磁盘上的方式，对数据的读写必须在所有盘上同时进行。因此，RAID2 和 RAID3 不适合多个并行 I/O 的响应。而 RAID4 到 RAID6 都采用了一种独立的存取方式，磁盘阵列中的每个磁盘的操作都是独立的，从而也就能同时响应多个 I/O 请求。

RAID4 又称为带有共享校验磁盘的独立数据磁盘系统(Independent Data Disks with Shared Parity Disk)。它与 RAID3 一样,同样采用“数据磁盘＋奇偶校验盘”的组织形式,如图 5-44 所示。但与 RAID3 不同的是,RAID4 不是以位为单位进行数据读写的。它将所有磁盘划分成大小相同的条带,每个条带能存储一个块的数据。

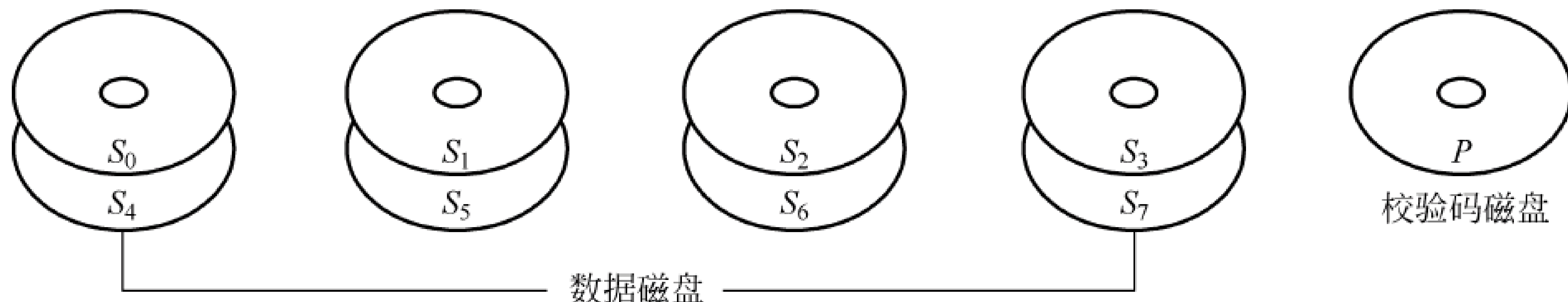


图 5-44　RAID4 的带奇偶校验磁盘的独立数据磁盘组织

校验方法是每次对每个数据盘上的对应条带的同一位进行奇偶校验运算生成这一位的奇偶校验位。对应一个数据条带也就生成一个奇偶校验条带。例如,假设某个由 4 张磁盘组成的 RAID4 磁盘系统的条带单位是字节,4 个磁盘上对应的一个条带分别为 S_0、S_1、S_2、S_3,与这一条带对应的偶校验条带为 P,则

$$P(1)=S_0(1)\oplus S_1(1)\oplus S_2(1)\oplus S_3(1)$$
$$P(2)=S_0(2)\oplus S_1(2)\oplus S_2(2)\oplus S_3(2)$$
$$\vdots$$
$$P(8)=S_0(8)\oplus S_1(8)\oplus S_2(8)\oplus S_3(8)$$

RAID4 的这种奇偶校验组织方式能有效地解决对并行 I/O 的响应问题。但它的写效率却较低。对于每一次写操作,磁盘阵列管理系统不仅要修改用户数据,而且要修改相应的奇偶校验位。对于上例来说,假设某次写操作只在 S1 条带上执行。

初始时,对每位 i 有下列关系式:

$$P(i)=S_0(i)\oplus S_1(i)\oplus S_2(i)\oplus S_3(i)$$

修改后,可能改变的位以撇号“′”表示:

$$\begin{aligned}P(i) &= S_0(i)\oplus S_1'(i)\oplus S_2(i)\oplus S_3(i)\\ &= S_0(i)\oplus S_1'(i)\oplus S_2(i)\oplus S_3(i)\oplus S_1(i)\oplus S_1(i)\\ &= S_0(i)\oplus S_1(i)\oplus S_2(i)\oplus S_3(i)\oplus S_1(i)\oplus S_1'(i)\\ &= P(i)\oplus S_1(i)\oplus S_1'(i)\end{aligned}$$

从上式可以看出,为了计算新的奇偶校验位,系统必须读取旧的数据条带和奇偶校验条带,然后用新的数据和新计算出的奇偶校验位修改上述两个条带。因此,每个条带的写操作包括两次读和两次写,使得写操作效率比较低。

从 RAID4 的构架和写操作可以看出,RAID4 的校验盘是整个系统的关键,它的失效会带来整个系统的失效。正因为这个原因,限制了 RAID4 的实际应用。

6. RAID5

RAID5 的全称是带分散校验盘的分条带数据磁盘系统(Striping with Floating Parity Drive),它是对 RAID4 加以改进的优化方案。对 RAID4 而言,校验盘是独立的,校验盘的失效将带来整个磁盘系统的失效,因此校验盘成为 RAID4 的瓶颈。RAID5 则是将奇偶检

验分散到整个磁盘阵列中，如图 5-45 所示。

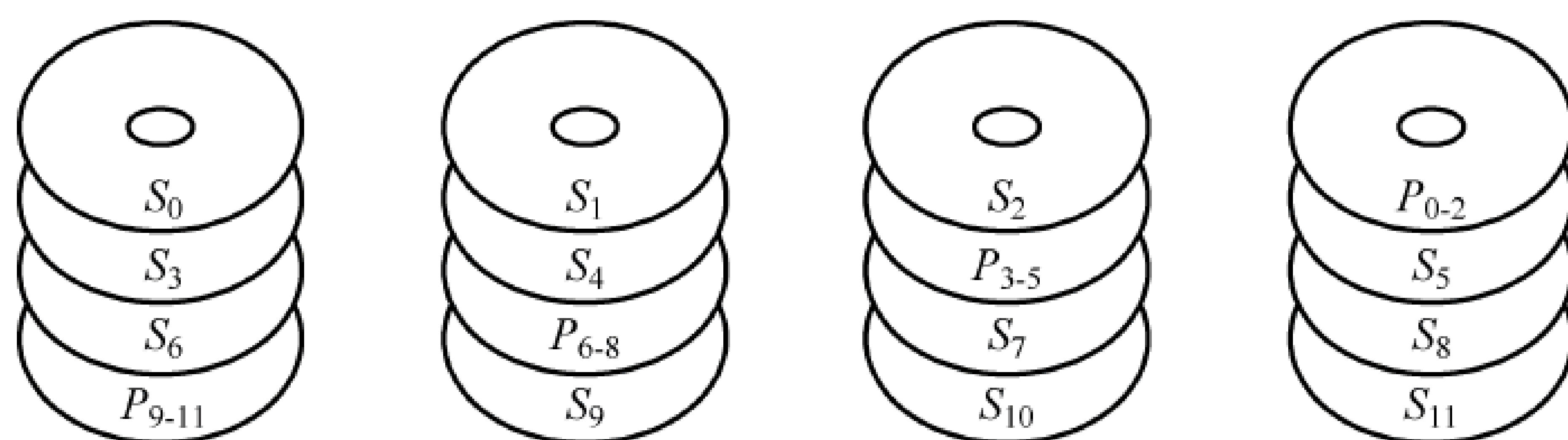


图 5-45 RAID5 的带分散校验盘的分条带数据磁盘组织

在图 5-45 中，对条带(S_0、S_1、S_2)、(S_3、S_4、S_5)、(S_6、S_7、S_8)、(S_9、S_{10}、S_{11})的校验条带 $P_{0\text{-}2}$、$P_{3\text{-}5}$、$P_{6\text{-}8}$ 和 $P_{9\text{-}11}$ 分布在不同的磁盘上。这样一方面保存了 RAID4 原有的独立磁盘所带来的并行 I/O 响应能力，另一方面又较好地解决了 RAID4 的校验盘瓶颈问题。但相对来说，RAID5 的磁盘控制功能是最复杂的。

RAID5 是目前在商业上得到了很好应用的 RAID 方案之一。

7. RAID6

前面所讨论的大多数 RAID 系统只能允许最多有一个磁盘失效。但实际情况是大型计算机系统的磁盘常常有发生成群成簇失效的现象。发生这种情况一般有两大原因：一是几乎同一时间生产的磁盘会在相同的时间里到达它们预期的使用寿命；二是磁盘的损坏通常是由于某些灾难性事件引起的，如电源故障等。如果在第一个损坏的磁盘还未来得及更换之前第二个磁盘又损坏了，那么整个磁盘系统将崩溃。RAID6 就是为解决这种问题而提出的一种方案。

RAID6 是由一些大型企业提出来的私有 RAID 级别标准，全称为带有两个独立分布式校验方案的独立数据磁盘(Independent Data Disks with Two Independent Distributed Parity Schemes)。它是在 RAID5 的基础上发展而成的，因此它的工作模式与 RAID5 有相似之处。所不同的是 RAID5 只使用了一种奇偶校验码，并只写入一个磁盘上；而 RAID6 除使用了奇偶校验码外，还使用了一种 Reed-Soloman 纠错编码来提供第二层保护，并将这两种校验码写入两个不同的磁盘中。这样就增强了磁盘的容错能力，允许磁盘阵列中出现故障的磁盘可以达到两个，图 5-46 是 RAID6 的组织架构图。

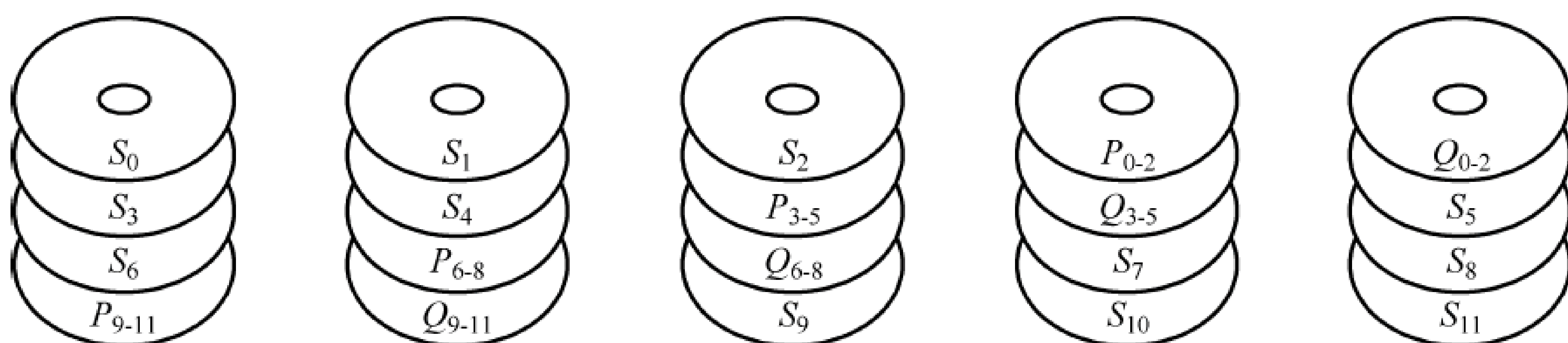


图 5-46 RAID6 的带有两个独立分布式校验方案的独立数据磁盘组织

虽然 RAID6 有更强的容错能力，但直到最近，还没有实际商业配置的 RAID6 系统。主要有两个方面的原因：一是生成 Reed-Soloman 编码需要相当大的费用代价；二是在 RAID6 系统中，更新磁盘上的纠错编码要求双倍的读写操作。

8. 混合 RAID 系统

许多大型的计算机系统并不局限于只使用一种类型的 RAID。在某些情况下,平衡磁盘系统的高可用性和经济性是非常重要的。例如,人们可能希望使用 RAID1 来保护包含操作系统文件等诸如此类重要数据,而对于用户数据文件使用 RAID5 就足够了。RAID0 非常适合于存放大程序运行过程中产生的临时文件,并且 RAID0 高效并行 I/O 访问特别适合于频繁磁盘操作的多进程运行。

用户还可以将多个 RAID 方案组合起来构建一种"新型"的 RAID。其中 RAID10 就是这样一种磁盘系统,它组合了 RAID0 的分带方式和 RAID1 的镜像功能。虽然代价昂贵,但是 RAID10 可以提供优良的读取性能和最佳的可用性。

最后,通过表 5-10 总结一下 RAID0～RAID6 的优缺点及适应的应用场合。

表 5-10　RAID0～RAID6 的优缺点及应用

RAID 级	优　　点	缺　　点	应　　用
RAID0	(1) 支持并行 I/O 的处理 (2) 没有因为校验、容错等而占用磁盘及 CPU 资源 (3) 使用和配置简单	(1) 缺乏校验恢复机制 (2) 没有磁盘数据容错能力 (3) 随磁盘数的增加可靠性降低	视频生成和图像处理以及其他并行 I/O 处理的频度较高的应用
RAID1	(1) 具有最佳失效保护功能 (2) 对磁盘的读取速度最快 (3) 使用和配置简单	占用的冗余磁盘最多	一些需要数据高可靠性的行业,如金融、政府部门等
RAID2	(1) 即时的数据校验,具有较好的数据恢复功能 (2) 并行数据传输速度快,但一次只能执行一个 I/O 请求	(1) ECC 占用比例较大 (2) ECC 的生成成本高且费时,导致整个磁盘阵列读写速度慢	目前的实际应用较少

续表

RAID 级	优　　点	缺　　点	应　　用
RAID3	(1) 简单高效的校验功能,具有较好的数据恢复功能 (2) 并行数据传输速度快,但一次只能执行一个 I/O 请求	磁盘控制器设计比较复杂	视频生成和图像处理以及其他需要较高数据传输率的应用
RAID4	(1) 高效率的数据校验功能 (2) 支持并行 I/O 的处理	(1) 磁盘控制器设计非常复杂 (2) 数据恢复及重建工作复杂 (3) 写效率低 (4) 校验盘成为系统瓶颈	在商业上没有得到应用
RAID5	(1) 高效率的数据校验功能 (2) 支持并行 I/O 的处理	磁盘控制器设计非常复杂	实际应用较广,包括应用于各种数据库服务器、文件和应用服务器等
RAID6	(1) 最强的磁盘容错功能,允许两个磁盘同时失效 (2) 支持并行 I/O 的处理	(1) 磁盘控制器设计非常复杂 (2) 写效率很低 (3) 校验码占用磁盘空间较多	目前的实际应用较少

知识拓展

网络存储系统

随着 Internet 的发展,网络上大量的数据需要存储,人们对更加简便、快速、安全地存储数据的需求对存储系统的容量和速度提出了空前的要求。传统的以服务器为中心的 DAS(Direct Attached Storage)方式(这种方式是将 RAID 硬盘阵列直接安装到网络系统的服务器上)已不能满足用户的需要,越来越多的用户已经从原来的以"服务器中心"模式转换为以"数据为中心"的网络存储模式上,这其中较为广泛使用的是 NAS 和 SAN。

NAS 的全称是 Network Attached Storage,中文翻译为网络直接存储,其构架如图 5-47(a)所示。在 NAS 存储结构中,存储系统不再通过 I/O 总线附属于某个特定的服务器或客户机,它完全独立于网络中的主服务器,可以看作一个专用的数据或文件服务器。NAS 包括处理器、磁盘系统和数据管理模块。NAS 和服务器、客户机一起连接成网络,客户机对数据的访问不再需要服务器的干预,允许客户机与 NAS 之间进行直接的数据访问。

SAN 的全称是 Storage Area Network,中文翻译为存储区域网络,其构架如图 5-47(b)所示。在 SAN 结构中,各种存储设备通过光纤及光纤通道交换机与服务器一起连接成高速网络,使与服务器连接成 LAN 的客户机可以非常方便地访问存储设备。在 SAN 结构中,每一个应用服务器上应安装文件管理系统,客户机对存储系统中的文件的访问必须通过服务器进行。

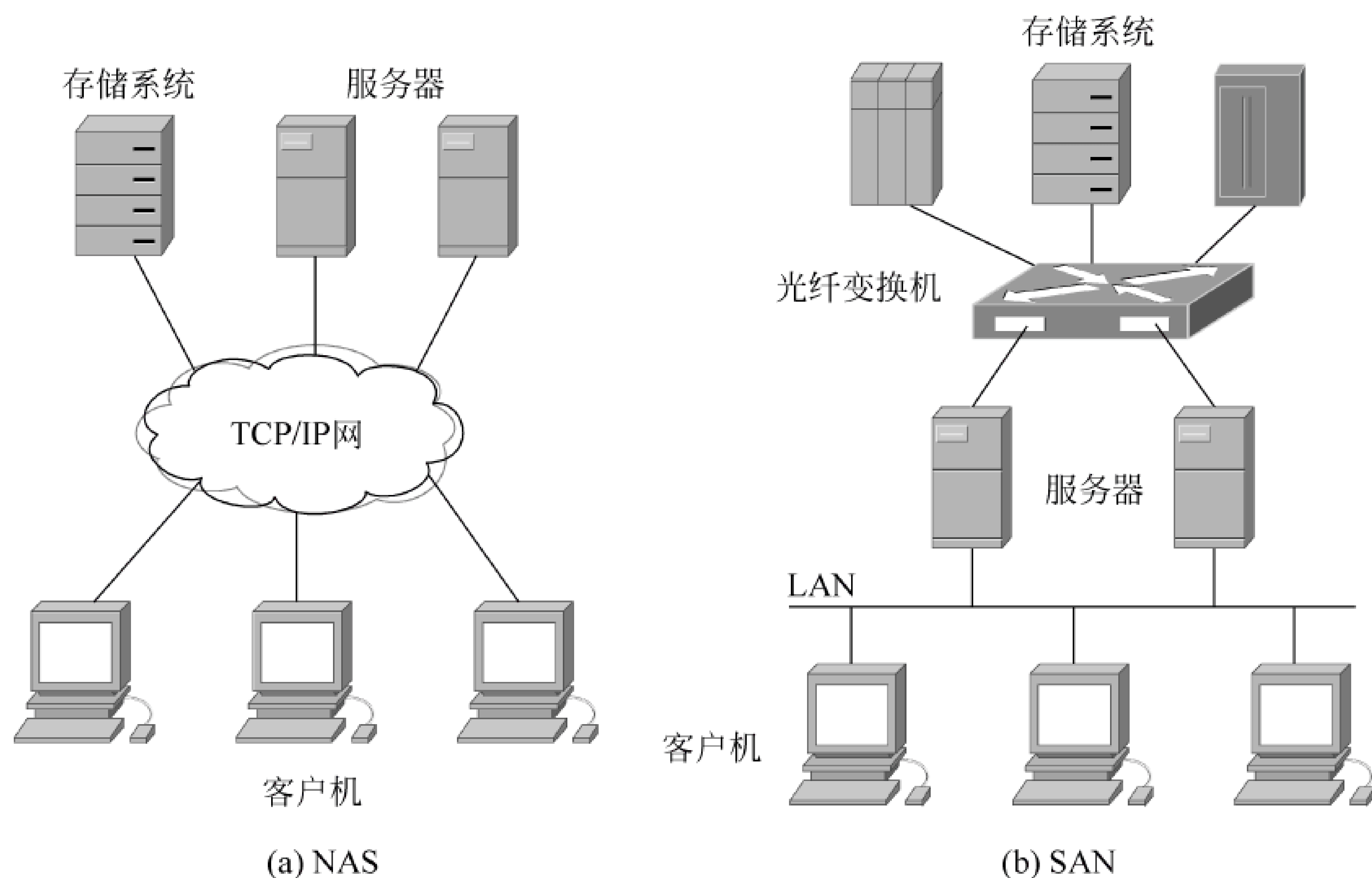


图 5-47 网络存储系统

本章小结

本章主要讲述了以下内容：

（1）输入输出接口的组成与功能，CPU 对输入输出设备的编址和管理。

（2）输入输出的传输控制方式，包括程序控制方式、中断控制方式、DMA 控制方式和通道方式等。这些不同的控制方式在 CPU 利用率、数据传输效率等方面各有不同，分别应用于对不同设备的输入输出控制。

（3）外部存储器的组织。介绍了磁盘存储器、磁带存储器和光盘存储器的数据组织及工作原理。

（4）RAID 技术。包括 RAID1～RAID6 的磁盘组织、校验及冗余技术的应用等。

习题五

一、名词解释

1. 外围设备	2. I/O 接口	3. I/O 端口	4. 中断
5. 中断向量	6. 单级中断	7. 多级中断	8. 中断嵌套
9. DMA	10. 通道	11. 磁道	12. 扇区
13. 磁盘存储密度	14. 磁盘访问时间	15. 磁盘数据传输率	16. RAID

二、选择题

1. 中断向量地址是（　　）。

A. 子程序入口地址　　B. 中断服务程序入口地址

C. 主程序地址　　D. 中断返回地址

2. 为了便于实现多级中断，保存现场信息最有效的方法是采用(　　)。

A. 通用寄存器　　B. 堆栈　　C. 辅存　　D. 通道

3. 如果有多个中断同时发生，系统将根据中断优先级响应优先级最高的中断请求，若要调整中断事件的中断处理次序，可以利用(　　)。

A. 中断嵌套　　B. 中断向量　　C. 中断响应　　D. 中断屏蔽

4. 在采用 DMA 方式高速传输数据时，数据传送是(　　)。

A. 在总线控制器发出的控制信号控制下完成的

B. 在 DMA 控制器本身发出的控制信号的控制下完成的

C. 由 CPU 执行的程序完成的

D. 由 CPU 响应硬中断处理完成的

5. 下列陈述中正确的是(　　)。

A. 在 DMA 周期内，CPU 不能执行程序

B. 中断发生时，CPU 首先执行入栈指令将程序计数器的内容保护起来

C. DMA 传送方式中，DMA 控制器每传送一个数据就窃取一个指令周期

D. 输入输出操作的最终目的是实现 CPU 与外设之间的数据传输

6. 一般来讲，主机与硬盘之间的数据交换比软盘快，其主要原因是(　　)。

A. 硬盘有多个盘片

B. 硬盘采用密封式安装

C. 硬盘使用的是铝片等“硬”基质，转速快

D. 硬盘容量更大

7. 以下可用作数据备份之用的光盘是(　　)。

A. CD-R　　B. CD-ROM　　C. CD-RW　　D. CD

8. 以下不具有校验恢复机制的 RAID 技术是(　　)。

A. RAID0　　B. RAID1　　C. RAID2　　D. RAID3

9. 下列有关 I/O 接口的叙述中错误的是(　　)。(2014 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

A. 状态端口和控制端口可以合用同一寄存器

B. I/O 接口中 CPU 可访问寄存器称为 I/O 端口

C. 采用独立编址方式时，I/O 端口地址和主存地址可能相同

D. 采用统一编址方式时，CPU 不能用访存指令访问 I/O 端口

10. 某设备中断请求的相应和处理时间为 100ns，每 400ns 发出一次中断请求，中断响应所允许的最长延迟时间为 50ns，则在该设备持续工作过程中 CPU 用于该设备的 I/O 时间占整个 CPU 时间百分比至少是(　　)。(2014 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

A. 12.5%　　B. 25%　　C. 37.5%　　D. 50%

11. 下列选项中，用于提高 RAID 可靠性的措施有(　　)。(2013 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

Ⅰ. 磁盘镜像　　Ⅱ. 条带化　　Ⅲ. 奇偶校验　　Ⅳ. 增加 cache 机制

A. 仅Ⅰ、Ⅱ　　B. 仅Ⅰ、Ⅲ　　C. 仅Ⅰ、Ⅲ和Ⅳ　　D. 仅Ⅱ、Ⅲ和Ⅳ

12. 某磁盘的转速为 10 000 转/分，平均寻道时间是 6ms，磁盘传输速率是 20MB/s，磁盘控制器延迟为 0.2ms，读取一个 4KB 的扇区所需的平均时间约为（　　）。(2013 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

A. 9ms　　B. 9.4ms　　C. 12ms　　D. 12.4ms

13. 下列关于中断 I/O 方式和 DMA 方式比较的叙述中，错误的是（　　）。(2013 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

A. 中断 I/O 方式请求的是 CPU 处理时间，DMA 方式请求的是总线使用权

B. 中断响应发生在一条指令执行结束后，DMA 响应发生在一个总线事务完成后

C. 中断 I/O 方式下数据传送通过软件完成，DMA 方式下数据传送由硬件完成

D. 中断 I/O 方式适用于所有外部设备，DMA 方式仅适用于快速外部设备

14. 响应外部中断的过程中，中断隐指令完成的操作除保护断点外，还包括（　　）。(2012 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

Ⅰ. 关中断　Ⅱ. 保存通用寄存器的内容　Ⅲ. 形成中断服务程序入口地址并送 PC

A. 仅Ⅰ、Ⅱ　　B. 仅Ⅰ、Ⅲ　　C. 仅Ⅱ、Ⅲ　　D. Ⅰ、Ⅱ、Ⅲ

三、综合题

1. I/O 接口一般由哪些部件组成？I/O 接口主要应实现哪些功能？
2. 对 I/O 端口的编址主要有哪两种方式？
3. 计算机都有哪些输入输出传输控制方式？这些方式各有何特点？分别应用于什么样的场合？
4. 计算机系统中的中断源是怎样分类的？都有哪些中断源？
5. 简述中断系统的中断处理过程。
6. 中断响应所实现的功能主要有哪些？
7. 简述采用中断向量法的中断响应过程。
8. 简述 DMA 的传输控制过程。
9. 通道与 DMA 有什么不同？
10. 通道主要有哪些种类？它们各有何特点？分别适合与什么样的设备连接？
11. 光盘主要有哪些种类？
12. 设某机器中断系统有 5 级中断 A、B、C、D、E，其中断响应优先级由高到低顺序为 A→B→C→D→E，现要求将中断处理次序改为 B→D→A→E→C，试问：

(1) 下表中各级中断处理程序的各中断屏蔽码如何设置（“1”表示屏蔽中断，“0”表示开放中断）？

(2) 若在执行一个 CPU 程序时这 5 级中断同时都发出中断请求，按更改后的次序画出各级中断响应和中断处理的过程示意图。

中断处理程序	中断屏蔽码				
	A	B	C	D	E
A					
B					
C					
D					
E					

13. 设某磁盘存储器转速为 7200r/min，平均找道时间为 10ms，每道存储容量为 600Kb，求磁盘的平均存取时间和数据传输率。

14. 设某磁盘存储器转速为 3000r/min，共有 18 个记录面，每面有 20 000 个磁道，每道存储容量为 300Kb，每道扇区数为 600，试求：

（1）磁盘存储器的总存储容量。

（2）磁盘数据传输率。

（3）磁盘的平均等待时间。

（4）每扇区的容量。

（5）给出一个磁盘地址格式方案。

15. 设某磁带机有 9 个磁道，带长 600m，带速 2m/s，每个数据块 1KB，块间隔 14mm，若数据传输率为 128 000B/s，试求：

（1）磁带的记录密度

（2）若带的首尾各空 2m，求磁带最大有效存储容量。

16. 有一台磁盘机，其平均找道时间为 30ms，平均等待时间为 10ms，数据传输率为 500KB/ms，磁盘机上存放着 1000 个文件，每个文件为 3000KB。现需要将所有文件逐个取走，更新后再放回原处（文件大小不变），已知对每个文件更新所需时间为 4ms，且更新操作与磁盘读写不相重叠。试问：

（1）更新磁盘上所有文件并重新写回磁盘共需多少时间？

（2）若磁盘机转速提高一倍，则更新磁盘上所有文件并重新写回磁盘共需多少时间？

17. RAID 有哪些分级？各有什么特点？

18. 设某单位要构建一个采用双机磁盘冗余阵列系统（使用 RAID1），试查阅资料，给出一个具体实现的方案。

19. 某计算机的 CPU 主频为 500MHz，CPI 为 5（即执行每条指令平均需 5 个时钟周期）。假定某外设的数据传输率为 0.5MB/s，采用中断方式与主机进行数据传送，以 32 位为传输单位，对应的中断服务程序包含 18 条指令，中断服务的其他开销相当于 2 条指令的执行时间。请回答下列问题，要求给出计算过程。（2009 年全国硕士研究生入学统一考试计算机学科专业基础综合试题）

（1）在中断方式下，CPU 用于该外设 I/O 的时间占整个 CPU 时间的百分比是多少？

（2）当该外设的数据传输率达到 5MB/s 时，改用 DMA 方式传送数据。假设每次 DMA 传送大小为 5000B，且 DMA 预处理和后处理的总开销为 500 个时钟周期，则 CPU 用于该外设 I/O 的时间占整个 CPU 时间的百分比是多少（假设 DMA 与 CPU 之间没有访存冲突）？

第6章 总线与接口组织

在计算机内部，CPU与存储器、输入输出接口之间需要互连，这种互连采用的是总线结构；在计算机外部，主机与输入输出设备以及主机与主机之间也需要互连，这种互连是使用标准总线接口进行的。

本章首先讲述计算机内部各部件之间的总线互连结构，介绍总线的基本概念、总线的类别和总线的控制方式等；然后列举几种现代微机中常用的总线标准，如ISA、PCI等；最后介绍几种目前在计算机中常用的外部总线接口标准，如USB、IEEE 1394和SCSI等。

6.1 互连结构

在第1章已介绍过，计算机从硬件上讲主要由CPU、存储器和输入输出接口等组成，它们不是孤立的部件，而是需要互相交换信息的，因此，它们之间需要通过某种方式互连。图6-1给出了CPU、存储器和输入输出接口需要交换的信息种类。

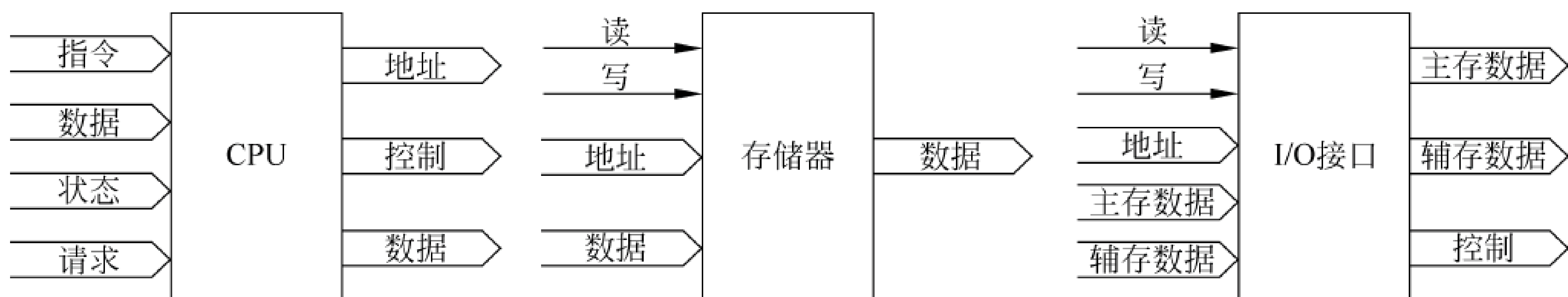


图6-1 CPU、存储器和I/O接口交换的信息种类

在计算机中，主要的通信包括以下几种。

（1）存储器到CPU：CPU从存储器中读指令或数据，为完成这一工作，CPU要向存储器给出地址信号并发出读控制信号。

（2）CPU到存储器：CPU向存储器中写入数据，为完成这一工作，CPU要向存储器给出地址信号和写入的数据，并发出写控制信号。

（3）I/O到CPU：CPU从I/O接口中读数据，为完成这一工作，CPU要向I/O接口给出地址信号并发出读控制信号，I/O接口向CPU提供设备状态信号和向CPU发出中断请求等控制信号。

（4）CPU到I/O：CPU向I/O接口中写入数据，为完成这一工作，CPU要向I/O接口给出地址信号和写入的数据，并发出写控制信号，I/O接口向CPU提供设备状态信号和向

CPU 发出中断请求等控制信号。

(5) I/O 与存储器之间：对于这种情况，I/O 接口代替 CPU 控制 I/O 设备与存储器之间进行直接数据传输(如 DMA 方式)。

对于多个部件之间的通信，可以有多种互连方式，如图 6-2 所示给出的是几种常见的部件互连结构。

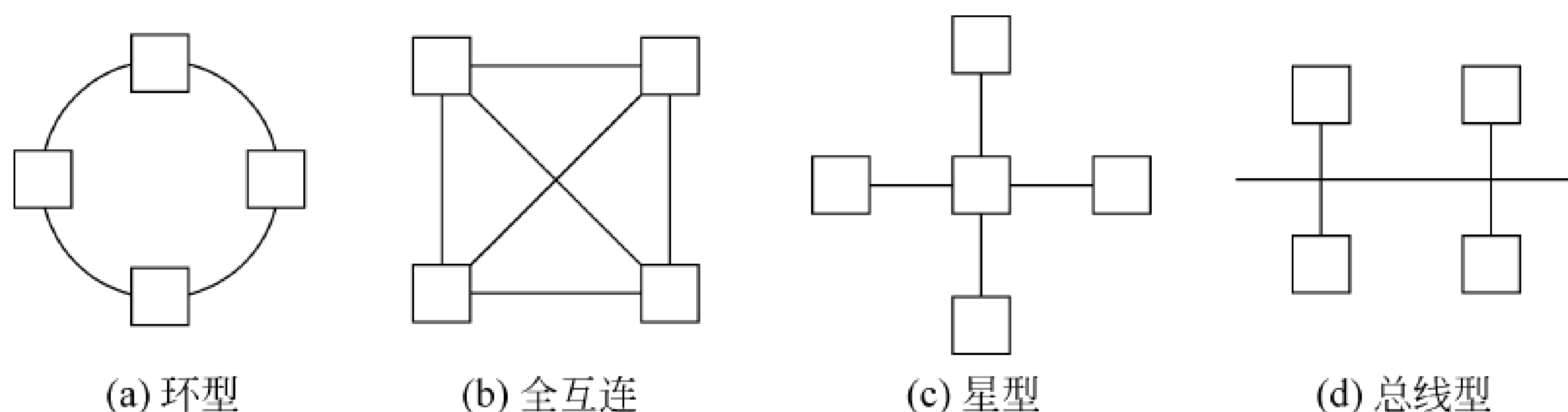


图 6-2 几种常见的部件互连结构

(1) 环型结构。所有部件连接成一个环状，环是所有部件通信的公共通路。这一结构的缺点是环的故障或任一部件的故障都将使整个系统通信瘫痪。

(2) 全互连结构。所有部件都一一相连，这种结构的优点是并行性好，两两部件之间可以同时通信。但缺点是使用的信号线多，每增加一个部件需要增加 n 组信号线，过多的信号线会使计算机系统的设计变得十分复杂。

(3) 星型结构。由一个核心部件与其他部件一一相连。这种结构虽然从理论上讲符合 CPU 作为计算机硬件核心与其他部件一一相连的模式，但实际上是不可行的，因为 CPU 每与一个部件相连就需要增加一组重复的引出信号线，这对 CPU 的设计来说是不现实的。另外，计算机系统是一个多主设备和多从设备互连的系统，若将 CPU 放置在核心位置，其他主设备就无法独立对从设备实施控制。

(4) 总线结构。这种方式是将所有部件连接在一组公共信号线上，总线是所有部件通信的公共通路。这一结构非常适合于计算机系统各部件的互连，其原因：一是连接结构简单，整个系统互连信号线最少；二是部件之间通过共享的总线进行通信并不影响并行性，因为在计算机系统中，更多的场合总是一个主设备与一个从设备进行通信，即使发生同时有多个主设备和多个从设备通信的情况，也可以通过合理的设计和调度较好地解决并行性问题。

多年来，人们通过尝试各种互连结构，迄今为止在计算机中普遍采用的是总线互连结构。

6.2 总线互连

总线是目前计算机中各部件互连的一种主要方式。本节主要介绍总线的基本概念、计算机中常见的总线互连结构和总线控制方式。

6.2.1 总线的基本概念

总线(Bus)是一组信号线，用于将两个或两个以上部件连接起来，形成部件之间的公共

通信通路，每个部件将自己的信号线与总线相连。

总线具有以下两个最主要的特点：

(1) 共享性。总线是供所有部件进行通信所共享的，任何两个部件之间的数据传输都是通过共享的公共总线进行的。

(2) 独占性。一旦有一个部件占用总线与另一个部件进行数据通信，其他部件就不能再使用总线，也就是说，一个部件对总线的使用是独占的。之所以这样，是因为计算机的总线是由一组传输电信号的信号线组成的，在任何时候一条信号线的有效状态或者为“0”或者为“1”(这种“0”、“1”状态是通过信号线的电压值来表示的)，若同时有两个部件向某条信号线发送数据，其中一个将信号线置“1”，另一个置“0”，这将导致信号线处于不稳定状态，也就是说，一条信号线无法同时接纳两个或两个以上部件对其状态的设置。

有趣的是，总线的英文单词与公共汽车(Bus)相同。实际上，可以将总线与公共汽车道路作一个对比。首先，相同之处：计算机中的总线是所有连接在其上的部件所发出信息传输的公共通路，而公共汽车道也是从所有站发出的公共汽车行驶的公共通路；不同之处：对计算机的总线而言，一旦有一个部件发出的信息在总线上传输，其他部件就不能再使用总线，这是由总线的特性所决定的；而对公共汽车道而言，同一时刻允许多个站发出的汽车在其上行驶。

从图 6-1 计算机各功能部件之间所交换的信息种类可以看出，总线中主要包含地址信息、数据信息和控制及状态信息等。因此，根据总线的功能划分，总线包括地址总线(Address Bus，AB)、数据总线(Data Bus，DB)和控制总线(Control Bus，CB)，另外，总线也提供电源线和接地线等。

(1) **地址总线**：地址总线上传送的是由 CPU 等主设备发往从设备的地址信号。当 CPU 对存储器或 I/O 接口进行读写时，首先必须给出所要访问的存储器单元的地址或 I/O 端口的地址，并在整个读写周期一直保持有效。

(2) **数据总线**：数据总线上传送的是各部件之间交换的数据信息。数据总线通常是双向的，即数据可以是由从设备发往主设备(称为读或输入)，也可以是由主设备发往从设备(称为写或输出)。

(3) **控制总线**：控制总线上传送的是一个部件对另一个部件的控制或状态信号，如 CPU 对存储器的读、写控制信号，外部设备向 CPU 发出的中断请求信号等。

总线按其所连接的不同部件或设备划分可分为以下几类：

1) 系统总线

在现代计算机系统中，CPU、存储器和部分输入输出接口都设计在机器的主板上，主板上还会设计一些 I/O 插槽(又称为 I/O 通道)用于接插一些 I/O 设备适配器，通过这些适配器连接 I/O 设备。用于将 CPU、存储器和输入输出接口及 I/O 通道连接起来的一组信号线就称为**系统总线**。系统总线是计算机系统最核心的一条总线，它的设计通常需符合一定的标准或规范，具有相同标准系统总线的机器通常在硬件上是兼容的。

计算机的系统总线通常具有三态性，即高电平状态(通常用于表示“1”)、低电平状态(通常用于表示“0”)和高阻状态。CPU 在正常工作状态下，其总线中的信号线或者处于高电平状态，或者处于低电平状态，而当 CPU 需要将总线的控制权交给其他 DMA 控制器等主设备时，它会将其连接的总线置为高阻状态。正是因为总线的这种三态性，使得多个主设备可

以同时连接在总线上。

一些实际系统总线的例子，如 ISA、EISA、MCA、STD、VME、MultiBus、PCMCIA、PCI 等。

2）局部总线

随着计算机对 CPU 访存速度和图像处理速度的要求越来越高，现代机器往往采用多总线结构设计，即将 CPU 与存储器及显示器适配器相连接的信号线从系统总线中分离出来，形成 CPU 与存储器和 CPU 与显示器适配器之间的专用总线，称为**局部总线**。

局部总线一般是 CPU 芯片引脚的延伸，与 CPU 密切相关。局部总线的设置可以大大提高 CPU 与存储器或显示器适配器之间的数据传输速率。

3）外部总线

外部总线主要是指计算机与计算机、计算机与外部设备之间的通信总线。一些已形成工业标准的外部总线除了对总线的信号线功能和通信规程等进行了定义外，还对机器之间或机器与设备之间的接口标准也做了规定。例如，用于微型计算机之间进行串行数据通信之用的 RS-232/RS-485 接口标准，微机与外部设备之间的 USB 和 IEEE 1394 通用串行总线接口标准等。

衡量总线性能的主要技术指标是总线带宽，它是指总线所能达到的最高数据传输率，单位是兆字节/秒(MB/s)。总线带宽在一定程度上影响到机器的整体性能，总线所能提供的最高数据传输率也在不断提高。例如微型计算机系统中采用的标准总线，其带宽从 ISA 总线的 8MB/s 发展到 EISA 总线的 33MB/s，PCI 总线的 133MB/s，PCI-X 总线的 1066MB/s 又发展到现在的 PCI-E 总线的 10GB/s 甚至更高。

6.2.2　总线互连结构

计算机各部件之间的总线互连结构随着计算机性能的不断提高也在不断发生着变化，这种变化主要体现在两个方面：一是由单总线结构向多总线结构发展；二是通过增加局部总线，提高 CPU 与一些部件之间数据交换的速度。

1. 单总线结构

单总线结构是指 CPU、存储器和各种设备接口之间通过单条总线相连，CPU 与存储器及各种设备接口之间的数据通信都是通过这条单总线进行的，如图 6-3 所示。单总线结构是早期计算机普遍采用的一种总线互连结构，在这种机器中，无论是快速还是慢速的各种设备的接口都和主存储器一起连接在同一总线上，因此对总线带宽的要求不能很高，这也就限制了 CPU 对一些相对高速的部件(如主存)的访问速度。

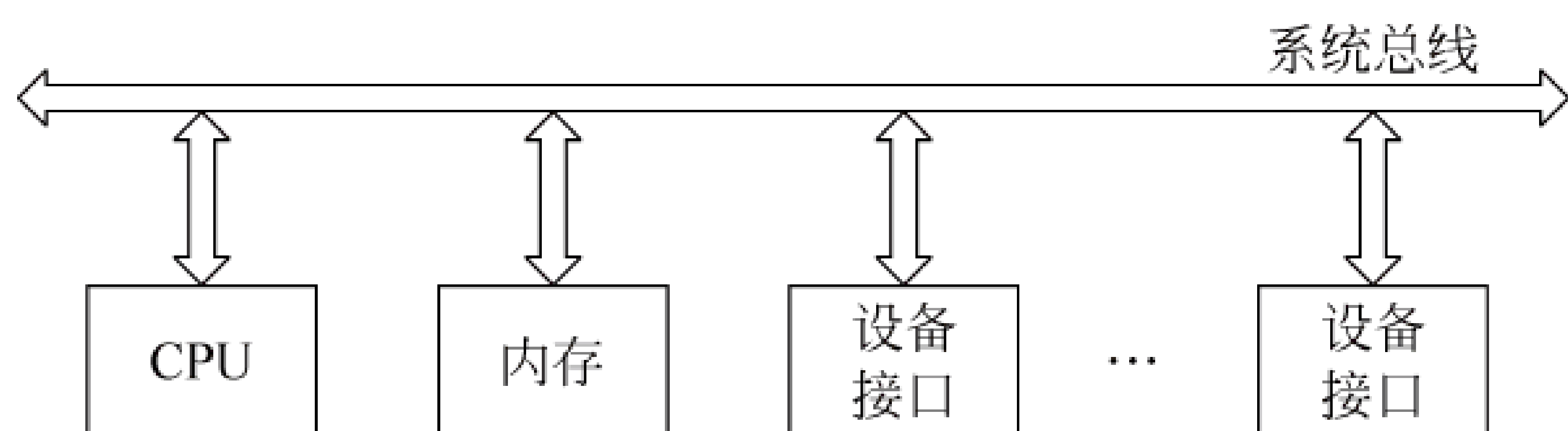


图 6-3　单总线结构

2. 局部总线结构

随着 CPU 性能的不断提高，CPU 与主存之间的速度差异越来越成为提高机器整体性能的瓶颈之一。为了减小这种差异，提高 CPU 的访存速度，计算机开始逐步使用高速缓存(cache)，并将之集成在计算机主板上。高速缓存的速度比主存速度更快，为充分发挥其作用，在 CPU 与高速缓存之间设置一条专用总线，又称存储总线或局部总线，它的总线带宽更宽，专用于 CPU 与高速缓存之间的数据交换。使用了局部总线的总线互连结构如图 6-4 所示。

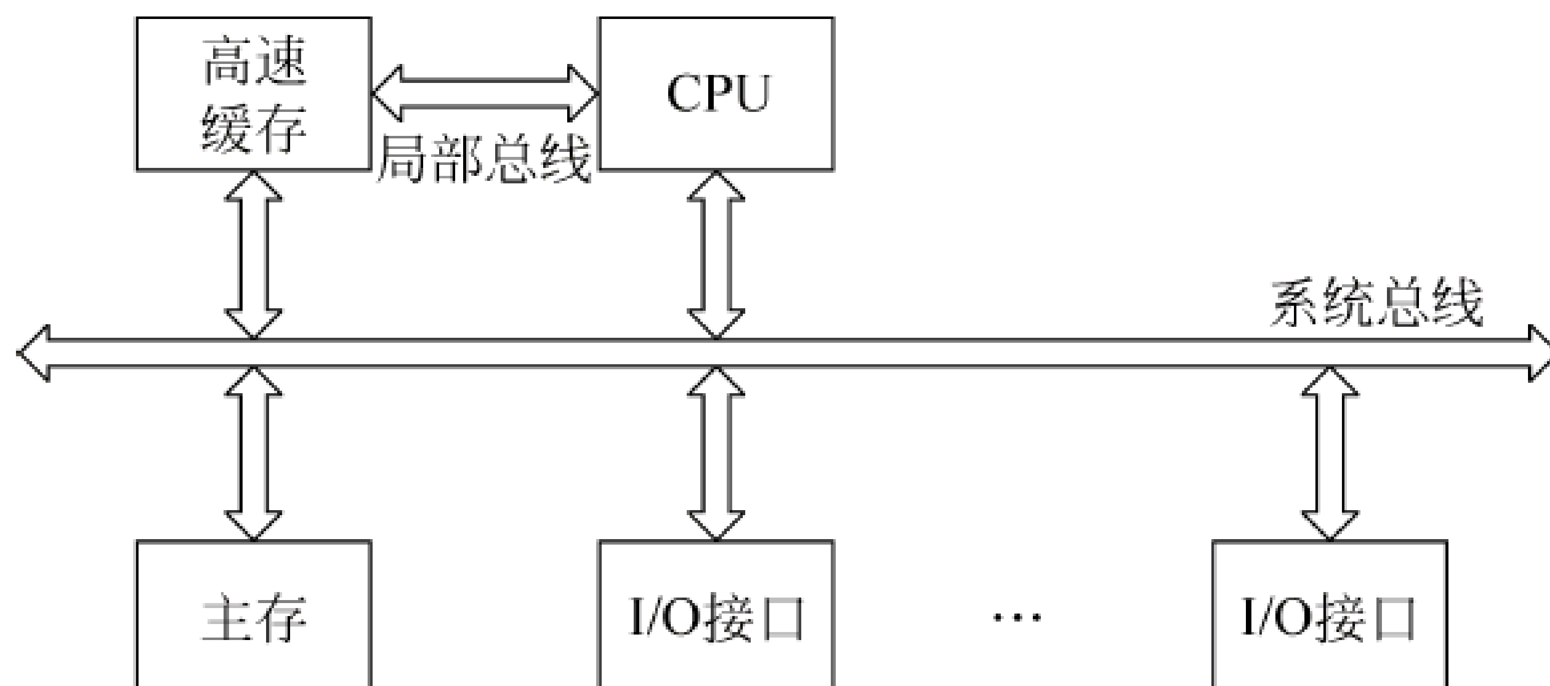


图 6-4 局部总线结构

3. 多总线结构

局部总线所具有的优势是明显的，同时它也为总线结构的发展奠定了良好的基础。在计算机中，不仅仅是存储器需要与 CPU 之间进行高速互连。多媒体技术的发展对计算机的图像处理速度提出了更高的要求；高速局域网的发展对计算机网络传输速度的要求越来越高；另外，CPU 与磁盘等高速辅存之间也同样需要高速数据交换，这就导致了在计算机系统中采用多总线结构。多总线结构的基本思想是：根据所连接部件或设备性能的不同，在机器系统中配置多条总线，原则上根据部件或设备的速度进行分类，将高速部件通过高速总线与 CPU 互连，低速部件通过低速总线与 CPU 互连。计算机系统的多总线结构如图 6-5 所示。

图 6-5 中，CPU 与主存之间通过传统的系统总线互连，而与高速缓存之间则使用一条专用高速存储总线连接；系统中的一些高速部件(如主机与 SCSI 硬盘的 SCSI 接口、显示接口和网络接口等)均连接在一条高速总线上，高速总线通过一个“桥”与系统总线相连；一些慢速设备则连接在低速总线上，低速总线通过另一个“桥”与高速总线相连。

6.2.3 总线的控制方式

1. 总线的仲裁

在第 5 章中介绍过，计算机系统的部件分为主设备和从设备，主设备是指能够通过总线控制和访问从设备的部件，如 CPU；而从设备是被主设备控制和访问的部件，如存储器。在机器系统中，有可能发生多个主设备同时申请使用总线的情况，但由于总线的特性决定了在同一时刻只允许一个主设备占用总线，因此，对于多个主设备同时申请总线的情况需要进行总线仲裁。

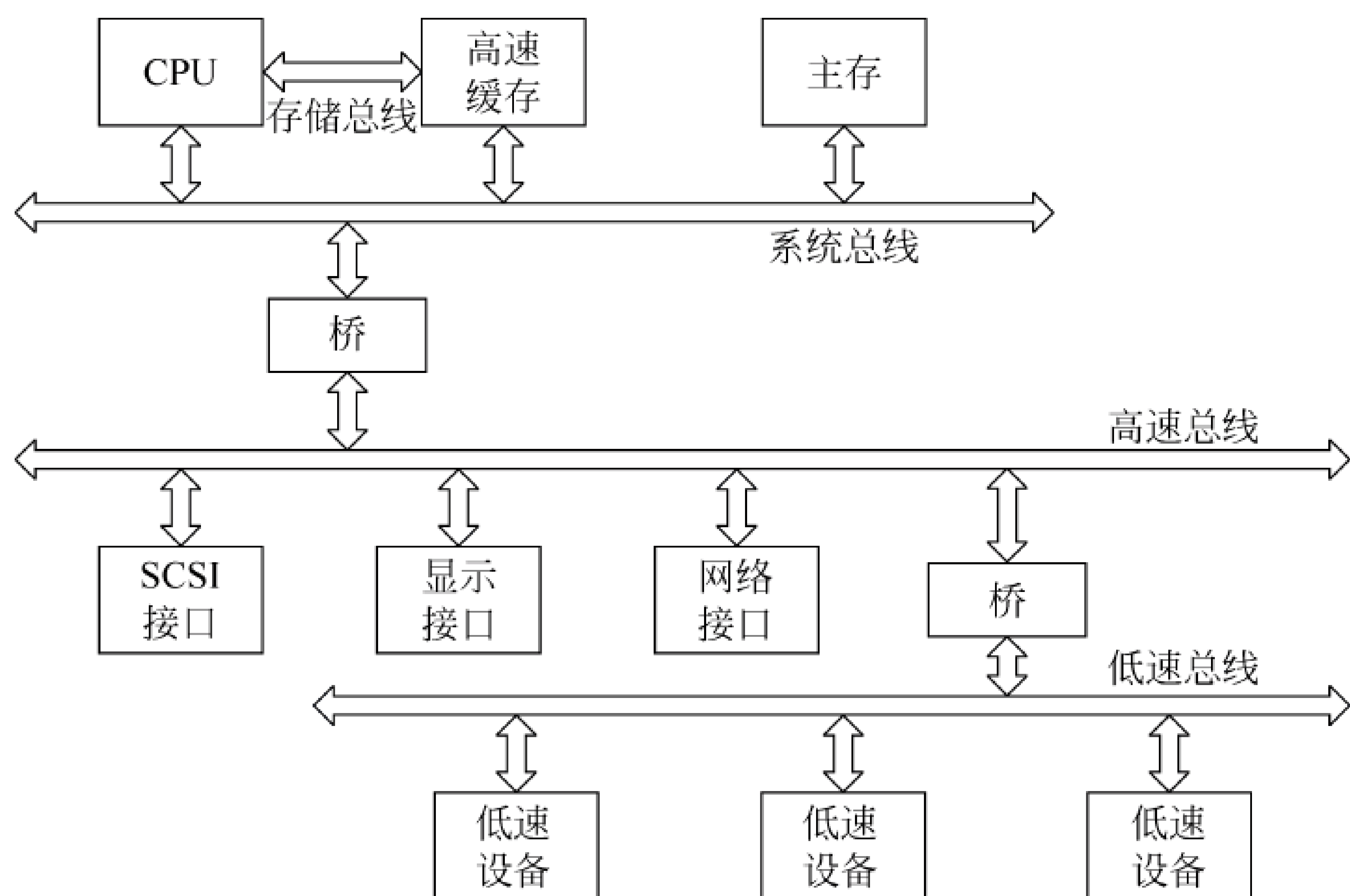


图 6-5　多总线结构

按照总线仲裁电路的位置不同，仲裁方式分为集中式仲裁和分布式仲裁两种。

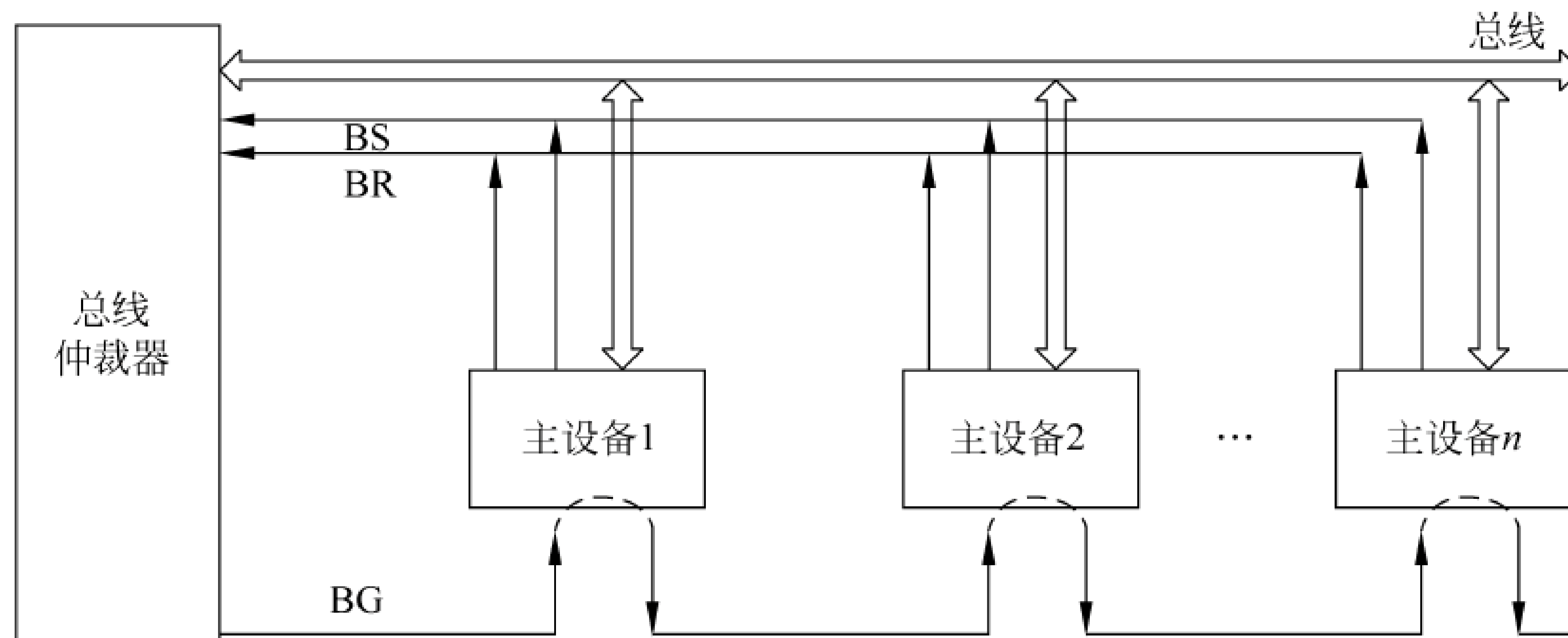
(1) 集中式仲裁。由一个称为总线控制器或仲裁器的硬件设备负责对多个主设备使用总线申请的裁决。总线仲裁器可以是独立的部件，也可以是 CPU 的一部分。常用的集中式仲裁方式主要有：链式查询方式、计数器定时查询方式和独立请求方式等。无论哪种方式，申请总线的主设备与总线仲裁器之间至少需要两条信号线的连接，一是总线请求信号线 BR，一是总线授权信号线 BG。

链式查询方式。如图 6-6(a)所示，总线授权信号 BG 串行地从一个设备传送到下一个设备，如果 BG 到达的设备无总线请求，则继续往下查询；如果 BG 到达的设备有总线请求，BG 信号就不再往下传，该设备就获得了总线使用权，建立总线忙信号 BS。链式查询方式的优点是只需很少几根线就能按一定优先次序实现总线仲裁，而且很容易扩充设备；缺点是对电路故障较敏感，如果第 i 个设备电路出现故障，那么第 i 个以后的设备都不能进行工作。另外链式查询方式不能保证公正性，即一个低优先级设备的请求可能长期得不到允许，因而优先级较低的设备可能长期不能使用总线。

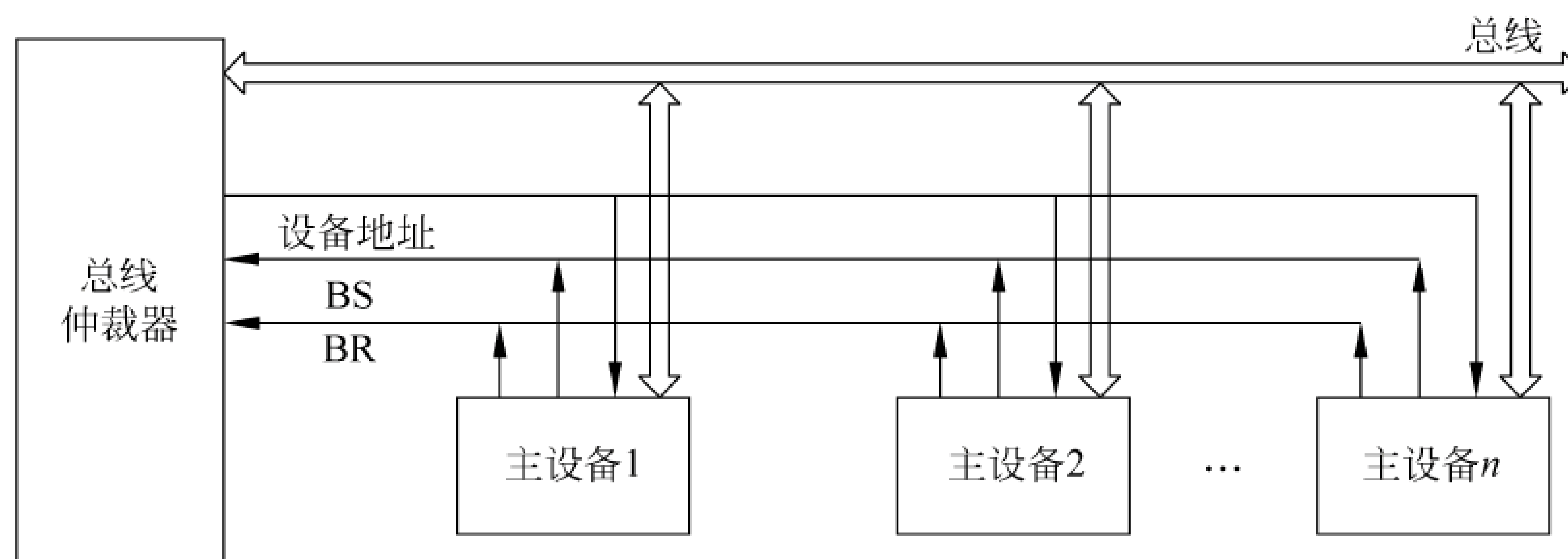
计数器定时查询方式。如图 6-6(b)所示，计数器定时查询方式比链式查询方式多了一组设备地址线，少了一根总线授权线 BG。总线上的任一设备要使用总线时，通过 BR 线发出总线请求，总线仲裁器收到总线请求信号后，在总线未被使用(BS＝0)的情况下让计数器开始计数，并将计数值通过一组地址线发往各设备，当地址线上的计数值与某个有总线请求的设备地址相一致时，该设备建立总线忙信号(BS＝1)，获得总线使用权，此时终止计数查询。计数器的初值可由程序设置，这就可以方便地改变设备的优先级。若每次计数都从“0”开始，此时各设备的优先次序与链式查询方式相同，优先次序是固定的；若每次计数的初值总是从上次的中止点开始，则每个设备使用总线的优先级就是相等的。计数器定时查询方式是以增加线数作为代价来换取灵活的优先次序的。

独立请求方式。图 6-6(c)给出的是一个采用独立请求方式的集中式总线仲裁电路图。每个设备都有一对总线请求线 BRi 和总线授权线 BGi。当设备要使用总线时，便发出总线请求信号，总线仲裁器按照一定的优先次序决定首先响应哪个设备的请求，然后发给设备总

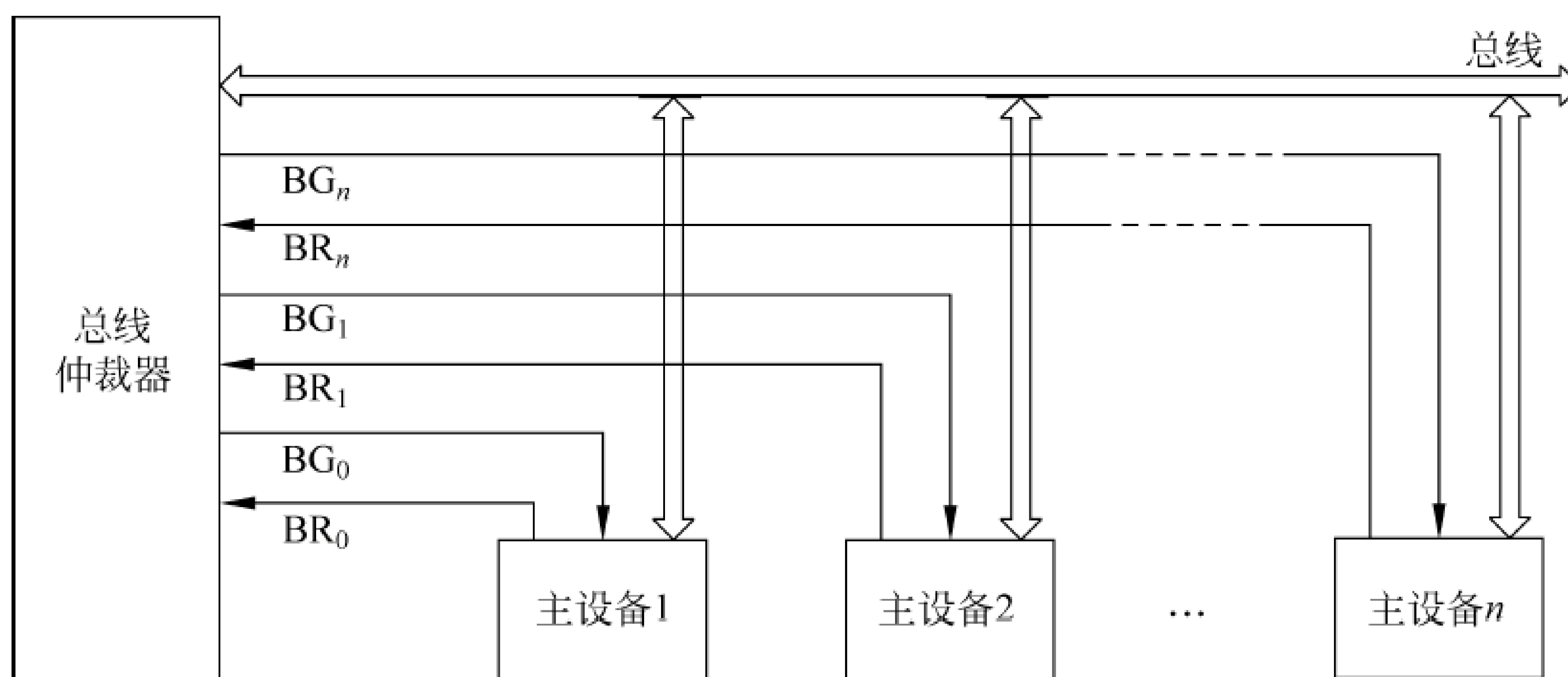
线授权信号 BGi。总线仲裁器可以给各个请求线以固定的优先级，也可以设置可编程的优先级。该方式的优点是响应速度快，缺点是控制线数量多。



(a) 链式查询方式



(b) 计数器定时查询方式



(c) 独立请求方式

图 6-6 集中式总线仲裁

假设用 n 表示允许挂接在总线上的最大设备数，则链式查询方式只需两根裁决线，计数器定时查询方式需要 $\log_2 n$ 根裁决线，而独立请求方式需要 $2n$ 根裁决线。

（2）分布式仲裁。总线仲裁电路分散于连接在总线上的各个主设备中。分布式仲裁不需要中央仲裁器，每个主设备都有自己的仲裁号和仲裁器。当它们有总线请求时，把它们唯一的仲裁号发送到共享的仲裁总线上，每个仲裁器将仲裁总线上得到的号与自己的号进行比较。如果仲裁总线上的号大，则它的总线请求不予响应，并撤销其仲裁号，最后，获胜者的

仲裁号保留在仲裁总线上。显然，分布式仲裁是以优先级仲裁策略为基础的。

2. 总线的定时

总线的定时是指为完成一次总线操作主、从设备所给出的地址、数据及控制信号在时序上的关系。总线的时序关系分为同步时序和异步时序。

1）同步时序

在同步时序下，总线中包含一条时钟信号线，总线上的所有操作都与时钟信号同步。时钟信号是一个由等宽的高、低电位交替出现的规则信号，一次高、低电位的交替称为一个**时钟周期**，每一个时钟周期中都含有一个上升沿和一个下降沿，总线上的操作正是通过由时钟周期的上升沿或下降沿触发来实现同步的。图 6-7 给出了一个典型的同步总线操作时序。

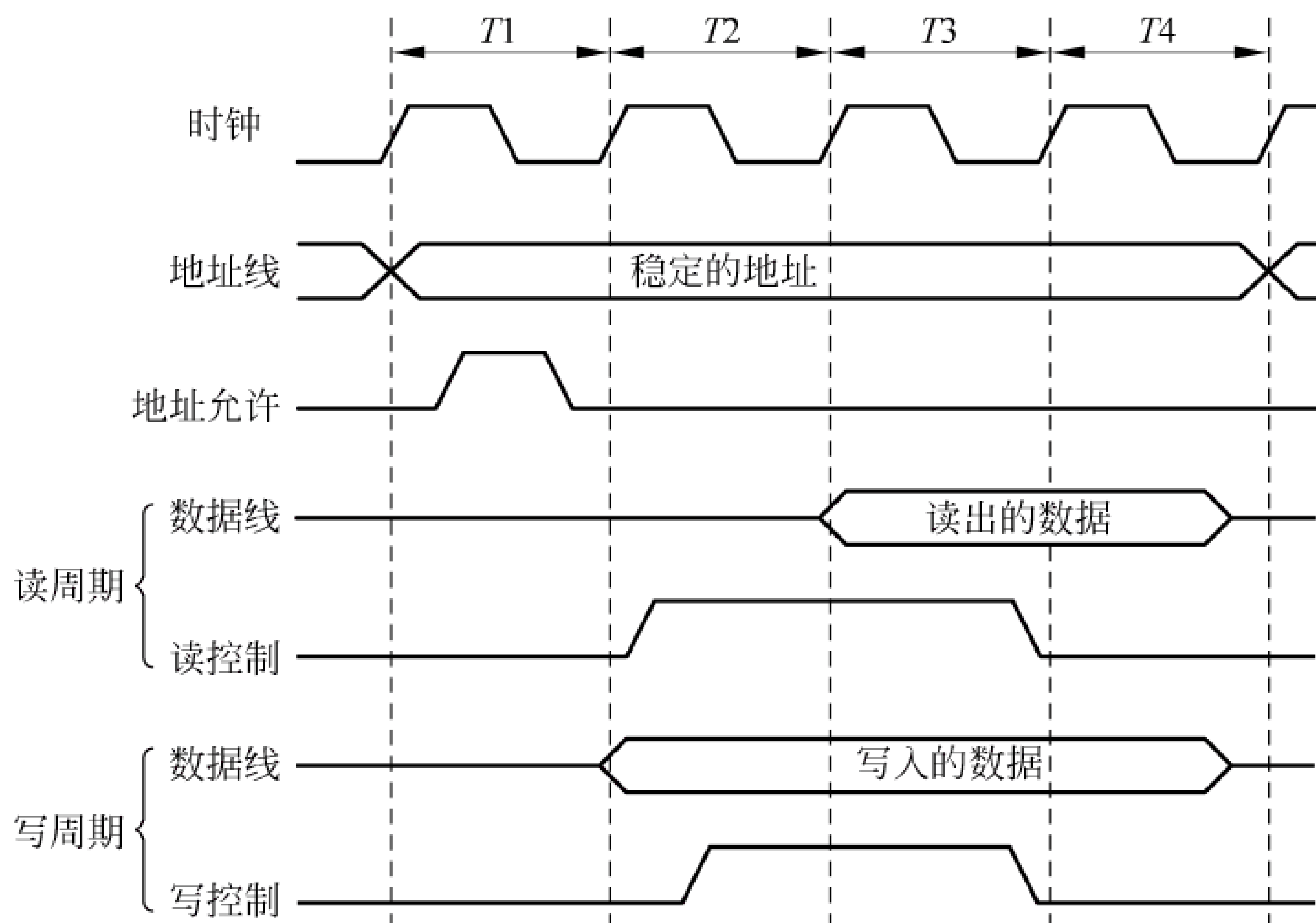


图 6-7　同步总线操作时序

图 6-7 中，一次总线操作需要 4 个时钟周期 $T1 \sim T4$，它们构成一个**总线周期**。主设备在第 1 个时钟周期的开始就给出所访问的从设备的地址信号，并保持在整个总线周期有效。若是读周期，则主设备需给出对从设备的读控制信号，经过一段时间后，从设备将数据读出到数据总线上。若是写周期，则主设备需给出对从设备的写控制信号，并将要写入的数据放到数据总线上，经过一段时间后写入从设备。所有地址、数据和控制信号都在时钟周期的上升沿或下降沿开始有效。

同步时序的优点是控制简单、实现容易；缺点是对各种操作来说总线周期往往是相同的，这对于短操作（所需时间短）来说，在时间上就有些浪费。

2）异步时序

在异步时序中，后一操作出现在总线上的时刻取决于前一操作的发生，即前后操作建立在应答式基础上，各种操作的产生不需要统一的公共时钟信号。图 6-8 给出了一个典型的异步总线操作时序。

异步时序的优点是总线周期长度可变，这对于快速和慢速的操作来讲都能做到高效；缺点是控制复杂，需要以更多的硬件成本为代价。

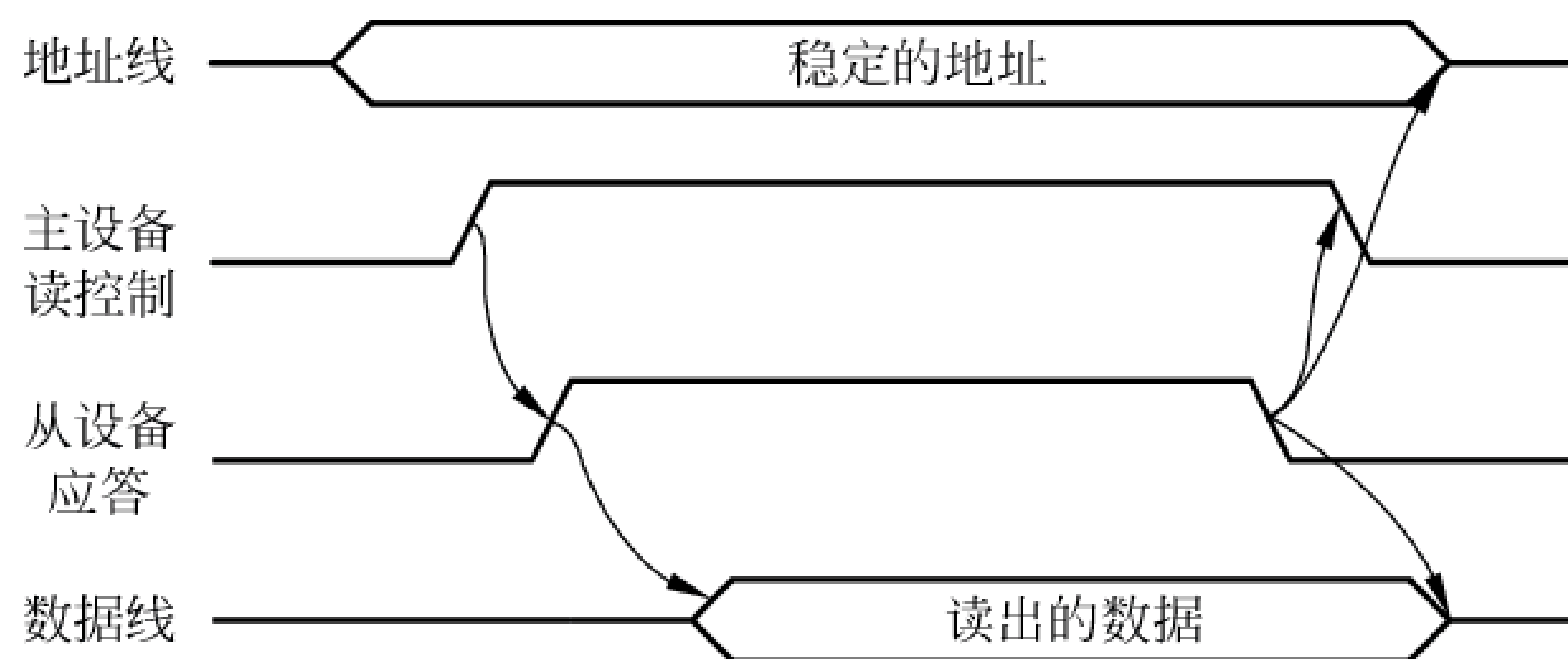


图 6-8 异步总线操作时序

3. 总线的数据传输

当代的总线标准大都能支持以下 4 种模式的数据传输。

1）单个字的读、写操作

单个字的读、写操作即一次总线操作完成一个字的读或写操作，字的宽度通常与总线宽度一致。读/写操作都是由主设备发起的，读操作是由从设备到主设备的数据传送；写操作是由主设备到从设备的数据传送。每次的字传输操作都是首先由主设备给出字单元地址，然后给出读/写控制信号。一次字的读/写操作一般在一个总线周期内完成。

2）块传送操作

块传送又称为猝发式传送。每次完成一个数据块的传输，每次传输主设备只需给出数据块的起始地址，然后对固定块长度的数据一个接一个地读出或写入，直到一个数据块全部传输完成。

3）写后读、读后写操作

每次操作只给出地址一次，然后对同一地址单元的内容或进行“先写后读”操作，或进行“读后写”操作。“先写后读”操作主要用于对如存储器的校验，即将一个数据先写入某一地址单元，然后读出比较，据此判断存储单元是否存在故障。“读后写”操作用于多道程序系统中对共享存储资源的保护。无论是“先写后读”还是“读后写”，整个操作都是不可分的。

4）广播操作

总线传送通常是在一个主设备和一个从设备之间进行，但有些总线允许一个主设备同时对多个从设备进行写操作，这种操作称为广播。

6.3 总线标准及举例

本节首先介绍总线标准包含了哪些方面的内容，然后举两个总线标准的例子，最后介绍在现代机器中总线的配置方式。

6.3.1 总线标准

不同厂家生产的部件或设备可以通过同一总线互连，例如同一厂家生产的显卡可以插接在不同厂家生产的计算机主机板上，而同一厂家生产的机器也可以插接不同厂家生产的显卡。之所以这样，是因为这些厂家生产的机器中的 I/O 插槽和显卡都遵循了同一个系统

总线标准。

总线标准是指芯片之间、部件之间、插板之间或系统之间通过总线进行连接和通信时应遵守的一些协议与规范。

通常一个总线标准应定义以下几个方面的特性(规范)。

(1) 物理特性：指总线的物理连接方式，包括总线信号线的条数，总线的插头、插座的形状和物理尺寸及引脚线的排列方式等。

(2) 电气特性：定义每一条线上信号的传递方向及有效电平范围。例如，TTL 高电平为 3.6～5V，低电平为 0～2.4V 等。若在总线上用高电平表示"1"、低电平表示"0"，则当某一部件需要通过某条信号线向另一部件传递"1"时，只需将这条线电压值置为 3.6～5V 即可，而接收的一方也同样使用的是 TTL 电平。

(3) 功能特性：描述总线中每一条线的功能，如定义地址线、数据线和各种控制及状态信号线。对系统总线而言，地址线的条数决定了 CPU 能访问的存储器的最大物理空间的大小；数据线的条数决定了 CPU 与存储器及设备间能够并行传输的数据的位数；而控制信号则根据 CPU 功能的不同而有所不同。一般来讲，总线中所包含的信号有以下几种。

① CPU 访存、访 I/O 的读写控制信号；

② 实现中断机制的中断控制信号；

③ 对其他主设备请求总线的总线仲裁信号；

④ 反映设备状态的状态信号；

⑤ 通信同步、复位、休眠等信号；

⑥ 电源和地线。

(4) 时间特性：即定义信号线之间的时序关系。机器中所有部件的操作都是在 CPU 产生的各种控制信号的控制下完成的，这些控制信号必须遵循一定的时序关系。换句话说，一个机器系统的设计正确与否，一方面取决于逻辑关系设计的正确与否，另一方面还取决于时序关系设计的正确与否。

6.3.2　ISA 总线

1981 年 IBM 公司推出了基于 8088 CPU 的 IBM PC 微型计算机，其上使用的是一个 62 条引脚的 XT 总线。1984 年 IBM 公司又推出了基于 80286 CPU 的 IBM PC/AT，其上使用的总线是 AT 总线。AT 总线是在原 XT 总线的基础上扩充而来的，总线引脚在原来 62 条的基础上又增加了 36 条，从而扩充到 98 条引脚。AT 总线具有 16 位数据宽度，最高工作频率为 8MHz，数据传输速率达到 16MB/s，地址线 24 条，可寻访 16MB 内存单元。1987 年 IEEE 以 XT/AT 总线为蓝本，定义了工业标准体系结构(Industry Standard Architecture，ISA)，其中原 XT 总线定义为 8 位 ISA 总线，原 AT 总线定义为 16 位 ISA 总线。ISA 总线在微型计算机上使用了二十多年，直到在以 P4 为 CPU 的主板上很少使用。图 6-9 给出了一个 ISA 总线在机器主板上的分布情况图。

ISA 总线具有很强的 CPU 相关性特征，也就是说，ISA 是针对某一 CPU 而专门设计的，其中 8 位 ISA 是针对 8088 CPU 设计的，而 16 位 ISA 是针对 80286 CPU 设计的。总线的引脚信号可以看成 CPU 引脚的延伸。例如 62 线的 ISA 总线中包含了与 8088 CPU 引脚相同的 20 条地址线、8 条数据线和读/写、中断请求、总线请求、复位及电源、地线等。而 16

位 ISA 增加的 36 线包括与 80286 CPU 相匹配的 8 条数据线、4 条地址线以及新增中断请求、总线请求线等。

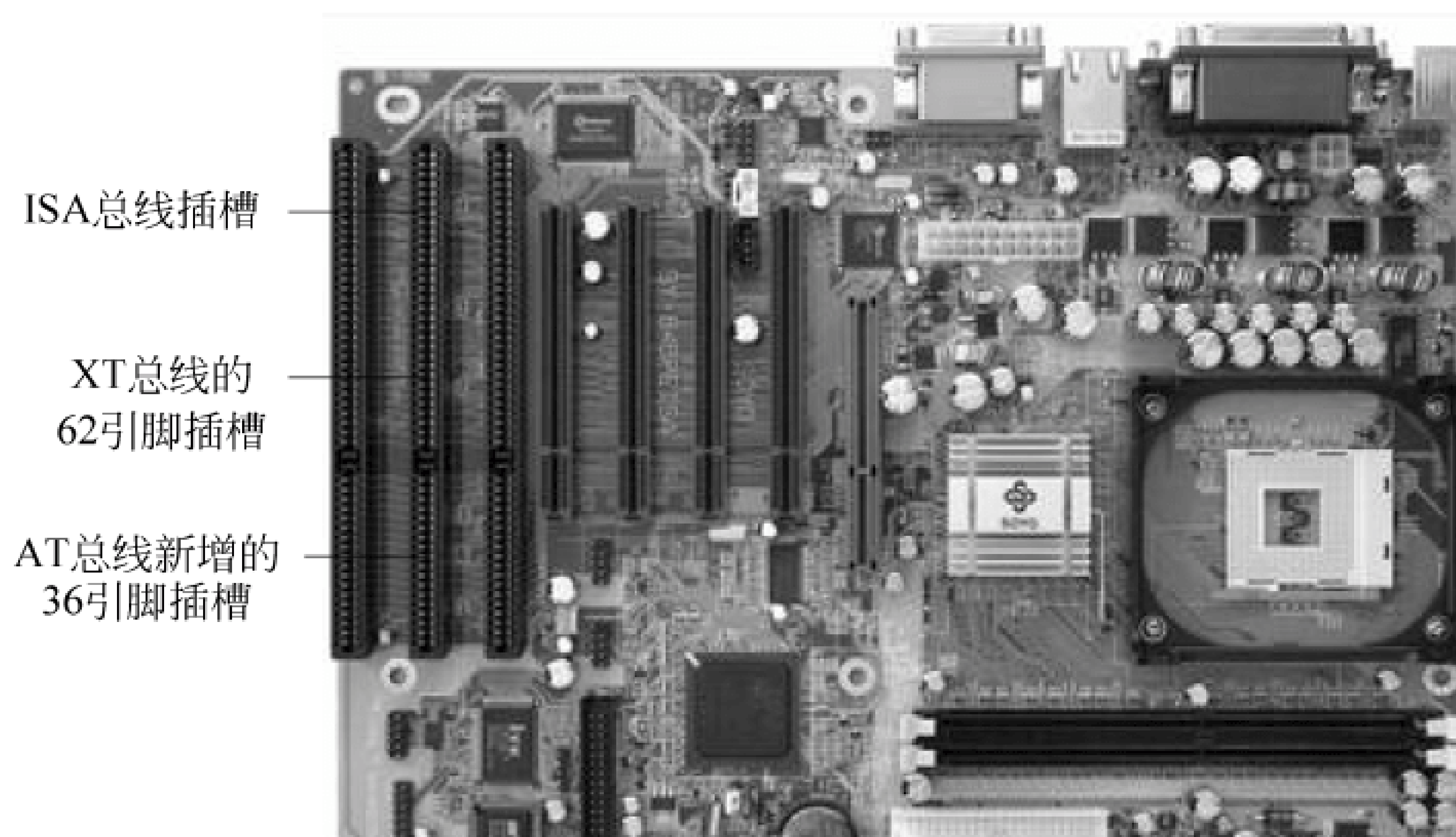


图 6-9 ISA 总线

ISA 总线在设计时采用的是单总线结构，主存和所有输入输出接口通过单条的 ISA 总线与 CPU 连接。

6.3.3 PCI 总线

随着 Windows 图形用户界面的迅速发展，以及多媒体技术的广泛应用，要求系统具有高速图形处理和 I/O 吞吐能力。为了适应计算机的这种发展要求，Intel 公司首先提出了 PCI(Peripheral Component Interconnect)总线的概念。之后 Intel 联合 IBM、Compaq、AST、HP、Apple、NCR、DEC 等一百多家公司共同开发总线，并于 1993 年推出了 PCI 总线标准。目前 PCI 已成为一种新的总线标准，广泛用于微机、工作站以及便携式计算机中。

1. PCI 总线的特点

PCI 总线主要具有以下一些特点。

1) 数据传输率高

PCI 的数据总线宽度为 32 位，并可扩充到 64 位。它以 33.3MHz 或 66.6MHz 的时钟频率工作，若采用 32 位数据总线，数据传输速率可达 133MB/s；而采用 64 位数据总线，则最高的数据传输速率可达 266MB/s。

2) 支持猝发传输

通常的数据传输是先输出地址后进行数据操作，即使所要传输数据的地址是连续的，每次也要有输出和建立地址的阶段。而 PCI 支持猝发数据传输周期，该周期在一个地址相位(phase)后可跟若干个数据相位。这意味着传输从某一个地址开始后，可以连续对数据进行操作，而每次的操作数地址是自动加 1 形成的。显然，这减少了无谓的地址操作，加快了传输速度。这种传输方式对使用高性能图形设备尤为重要。

3) 支持多主设备

在同一条 PCI 总线上可以有多个主设备，各个主设备通过总线仲裁竞争总线控制权。

相比之下，在 ISA 总线系统中，DMA 控制器和 CPU 对总线的争用是不平等的，DMA 控制器采用“周期窃取”法向 CPU 申请总线，得到 CPU 的允许后才能使用总线。而 PCI 总线专门设有总线占用请求和总线占用允许信号，各个主设备平等竞争总线。

4）独立于处理器

传统的系统总线（如 ISA 总线）实际上是 CPU 引脚信号的延伸或再驱动，而 PCI 总线以一种独特的中间缓冲器方式独立于处理器，并将 CPU 子系统与外围设备分开。一般来说，在 CPU 总线上增加更多的设备或部件，会降低系统性能和可靠程度。而有了这种缓冲器的设计方式，用户可随意增添外围设备，而不必担心会导致系统性能的下降。这种独立于 CPU 的总线结构还可保证外围设备互连系统的设计不会因处理器技术的变化而变得过时。

5）支持即插即用

所谓即插即用（Plug and Play），是指在新的接口卡插入 PCI 总线插槽时，系统能自动识别并装入相应的设备驱动程序，因而立即可以使用。即插即用功能使用户在安装接口卡时不必再拨开关或设跳线，也不会因设置有错而使接口卡或系统无法正常工作。即插即用的硬件基础是每个 PCI 接口卡（PCI 设备）中的 256 个字节的配置寄存器。在操作系统启动时或在 PCI 接口卡刚接入时 PCI 总线驱动程序要访问这些寄存器，以便对其初始化，并装入相应的设备驱动程序。

6）适用于多种机型

PCI 总线适用于各种类型的计算机系统，如台式计算机、便携式计算机及服务器等。

通过支持 3.3V 的电源环境，PCI 总线可应用于便携式计算机中。在服务器环境下，往往要求能连接较多的外围设备，而 PCI 总线规则规定一个计算机系统中可同时使用多条 PCI 总线，这又使得 PCI 总线非常适合应用于服务器中。

2. PCI 总线的系统结构

PCI 总线属于局部总线，它支持多总线结构，允许 PCI 总线和 CPU 总线及其他总线在同一个机器系统中并存。它的这一特点使得采用 PCI 总线的机器系统的设计非常灵活，一方面可以通过 PCI 总线连接各种 PCI 设备，另一方面可以通过其他总线连接一些非 PCI 设备。图 6-10 给出了一个采用 PCI 总线的机器系统的结构图。

从图 6-10 中可以看出，连接在 CPU 总线上的 cache 和主存系统经过一个桥与 PCI 总线相连。此桥电路提供了一个低延迟的访问通路，从而使 CPU 能够直接访问通过它映射于存储器空间或 I/O 空间的 PCI 设备；也提供了能使 PCI 主设备直接访问主存的高速通路。该桥电路还能提供数据缓冲功能，以使处理器与 PCI 总线上的设备并行工作而不必相互等待。另外，桥电路可使 PCI 总线的操作与 CPU 局部总线分开，以免相互影响，实现了 PCI 总线的独立驱动控制。

扩展总线桥电路的设置是为了能在 PCI 总线上引出一条传统的标准 I/O 扩展总线，如 ISA、EISA 或 MCA 总线，从而可继续使用原有的 I/O 设备，以增加 PCI 总线的兼容性和选择范围。通常，典型的 PCI 局部总线最多支持三个插槽。如果一个系统中要连接多个 PCI 设备，可通过增加桥电路的方式形成多条 PCI 局部总线。在一个系统中最多可有 256 条 PCI 局部总线。通常，把连接 CPU 总线与 PCI 总线的桥电路称为主桥或宿主桥（Host Bridge），连接 PCI 总线与 PCI 总线的桥电路称为 PCI-PCI 桥，连接 PCI 总线与其他总线的

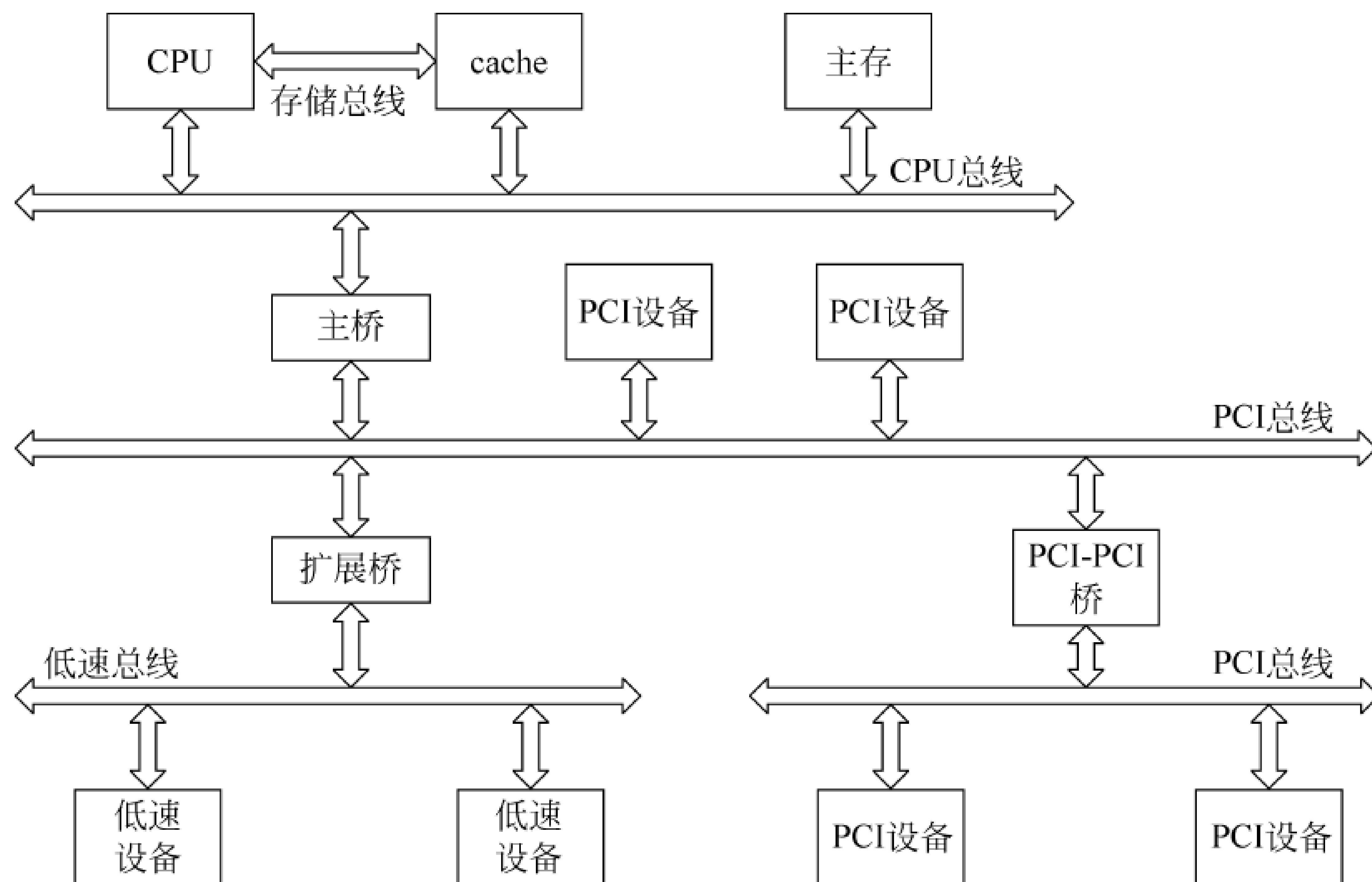


图 6-10 PCI 总线结构

桥电路称为 PCI 扩展桥。

3. PCI 总线的信号定义

在一个 PCI 系统中，总线信号通常分为必备和可选两大类。如果只作为从设备，则至少需要 47 根信号线；若作为主设备，则需要 49 根信号线。另外还有可选信号线 51 条，主要用于 64 位扩展、中断请求、高速缓存支持等。PCI 信号线总数为 120 条(包括电源、地、保留引脚等)。PCI 总线信号按功能分组的表示如图 6-11 所示。其中信号名称后面有“#”号表示低电平有效，否则表示高电平有效；对于有两种意义的信号(如 C/BE[0]#)，低电平时表示有“#”号的信号(例中的 BE[0])起作用，高电平时表示没有“#”号的信号(例中的 C)起作用。“#”相当于通常表示中的上横杠。

(1) 系统信号。

CLK：输入，系统时钟信号，为所有 PCI 传输提供时序，对于所有的 PCI 设备都是输入信号。其频率最高可达 33MHz/66MHz，这一频率也称为 PCI 的工作频率。

RST#：输入，复位信号，用于将所有 PCI 专用的寄存器、定序器和信号设置为初始状态。

(2) 地址和数据信号。

AD[31-0]：这是一组双向三态的 32 位地址、数据的复用信号。PCI 总线上地址和数据的传输必须在 FRAME# 有效期间进行，在 FRAME# 有效时的第 1 个时钟，AD[31-00]上的信号为地址信号，称为地址期；当 IRDY# 和 TRDY# 同时有效时，AD[31-00]上的信号为数据信号，称为数据期。一个 PCI 总线传输周期包含一个地址期和接着的一个或多个数据期。

C/BE[3-0]#：总线命令和字节允许复用信号，双向三态。在地址期，这 4 条线上传输的是总线命令；在数据期，它们传输的是字节允许信号，用来指定在数据期 AD[31-00]线上

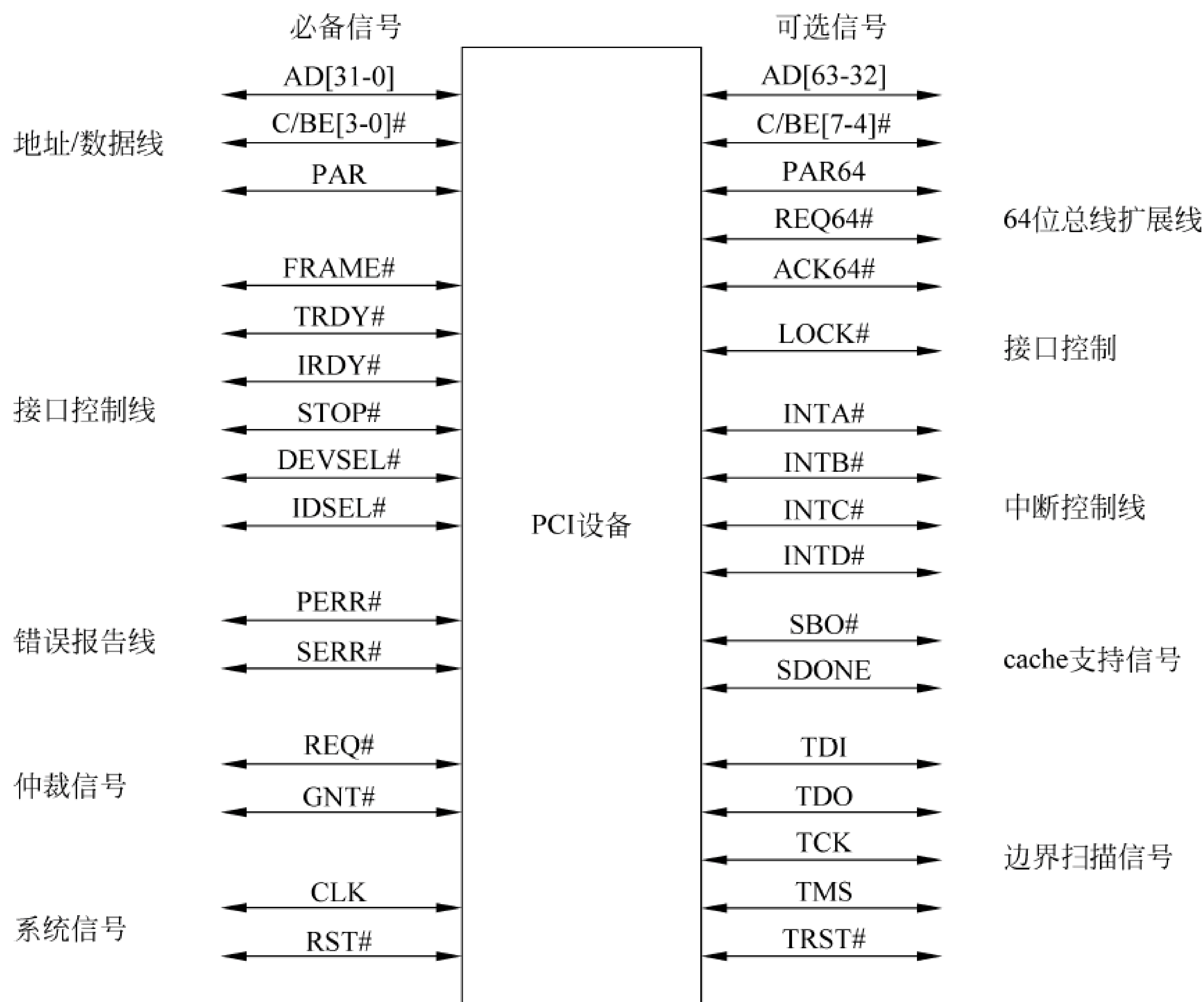


图 6-11　PCI 总线信号

4 个数据字节中哪些字节为有效数据。

PAR：奇偶校验信号，双向三态。它通过 AD[31-0]和 C/BE[3-0]进行奇偶校验。主设备在地址周期和写数据周期驱动 PAR，从设备在读数据周期驱动 PAR。

（3）接口控制信号。

FRAME＃：帧周期信号，双向三态，由主设备驱动，表示一次总线传输的开始和持续时间。当 FRAME＃有效时，预示总线传输的开始；在其有效期间，先传地址，后传数据；当 FRAME＃撤销时，预示总线传输结束，并在 IRDY＃有效时进行最后一个数据期的数据传送。

IRDY＃：主设备准备好信号，双向三态。IRDY＃要与 TRDY＃联合使用，当两者同时有效时，数据方能传输，否则，即为未准备好而进入等待周期。在写周期，该信号有效时，表示数据已由主设备提交到 AD[31-0]线上；在读周期，该信号有效时，表示主设备已做好接收数据的准备。

TRDY＃：从设备(被选中的设备)准备好信号，双向三态。同样 TRDY＃要与 IRDY＃联合使用，只有两者同时有效，数据才能传输。

STOP＃：从设备要求主设备停止当前的数据传送的信号，双向三态。显然，该信号应由从设备发出。

LOCK＃：锁定信号，双向三态。当对一个设备进行可能需要多个总线传输周期才能完成的操作时，使用锁定信号 CLK，进行独占性访问。例如，某一设备带有自己的存储器，那么它必需能进行锁定，以便实现对该存储器的完全独占性访问。也就是说，对此设备的操

作是排他性的。

IDSEL：输入，初始化设备选择信号。在参数配置读/写传输期间，用作片选信号。

DEVSEL# S/T/S：设备选择信号，双向三态。该信号由从设备在识别处地址时发出，当它有效时，说明总线上有某处的某一设备已被选中，并作为当前访问的从设备。

(4) 仲裁信号(只用于总线主控器)。

REQ#：总线占用请求信号，双向三态。该信号有效表明驱动它的设备要求使用总线。它是一个点到点的信号线，任何主设备都有它自己的 REQ# 信号。

GNT#：总线占用允许信号，双向三态。该信号有效，表示申请占用总线的设备的请求已获得批准。

(5) 错误报告信号。

PERR#：数据奇偶校验错误报告信号，双向三态。一个设备只有在响应设备选择信号(DEVSEL#)和完成数据期之后，才能报告一个 PERR#。

SERR#：漏极开路信号，低电平有效。系统错误报告信号，用作报告地址奇偶错、特殊命令序列中的数据奇偶错，以及其他可能引起灾难性后果的系统错误。它可由任何设备发出。

(6) 中断信号。

INTx#：漏极开路信号，电平触发，低电平有效的中断请求信号。此类信号的建立与撤销与时钟不同步。对于单功能设备，只有一条中断线，而多功能设备最多可有 4 条中断线。在前一种情况下，只能使用 INTA#，其他 3 条中断线没有意义。所谓多功能的设备是指将几个相互独立的功能集中在一个设备中。PCI 总线中共有 4 条中断请求线，分别是 INTA#、INTB#、INTC# 和 INTD#，均为漏极开路信号，其中后三个只能用于多功能设备。一个多功能设备上的任何功能都可对应于四条中断请求线中的任何一条，即各功能与中断请求线之间的对应关系是任意的，没有附加限制。两者的最终对应关系由中断引脚寄存器定义，因而具有很大的灵活性。如果一个设备要实现一个中断，就定义为 INTA#；要实现两个中断，则定义为 INTA# 和 INTB#，其他情况以此类推。对于多功能设备，可以多个功能公用同一条中断请求线，或者各自占一条，或者是两种情况的组合；而单功能设备，只能使用一条中断请求线。

(7) cache 支持信号。

为了使具有缓存功能的 PCI 存储器能够和全写式(Write-through)或回写式(Write-back)的 cache 操作相配合，PCI 总线设置了两个高速缓冲支持信号。

SBO#：窥视返回信号，双向低电平有效。当该信号有效时，表示命中了一个修改行。

SDONE#：查询完成信号，双向低电平有效。当它有效时，表示查询已经完成；反之，查询仍在进行中。

(8) 64 位扩展信号。

REQ64#：64 位传输请求信号，双向三态，低电平有效。该信号用于 64 位数据传输，由主设备驱动，时序与 FRAME# 相同。

ACK64#：64 位传输响应信号，双向三态，低电平有效，由从设备驱动。该信号有效表明从设备将启用 64 位通道传输数据，其时序与 DEVSEL# 相同。

AD[63～32]：扩展的 32 位地址和数据复用线。

C/BE[7～4]#：高32位总线命令和字节允许信号。

PAR64：高32位奇偶校验信号，是AD[63～32]和C/BE[7～4]的校验位。

PCI总线使用124线总线插槽，用于连接总线板卡，板卡的总线连接头上每边各有62个引线。扩充到64位时，总线插槽需增加64线，变成188线。相应地，板卡的总线连接头上每边变成94线。限于篇幅，这里不列举PCI总线插槽及板卡的全部188个引脚，读者如有需要，可查阅有关资料。

6.3.4　PCI-X 总线

PCI-X是由IBM、HP、Compaq提出来的，它是并行接口，是PCI的修正，也就是兼容PCI。PCI-X是在增加了电源管理功能和热插拔技术的PCI V2.2版本的基础上，将PCI的总带宽由133MB/s增至1.066GB/s。同时它还采用了分离事务即多任务的设计，允许一个正在向某个目标设备请求数据的设备，在目标设备未准备好之前处理其他任何事情；而在目前的PCI体系中，设备在完成一次请求之前不能理会任何事情，此时的总线时钟周期都被白白浪费掉了。同时PCI-X还允许把没有准备好发送数据的设备从总线上移走，这样总线带宽可以被其他事务使用，使总线的利用率大幅上升。所以，在相同的频率下，PCI-X将能提供比PCI高14%～35%的性能。PCI-X还采用了与IA-64相同的128b标准尺寸数据块设计，使通过总线的数据块大小相同，这样就提供了更多的流水线机制，改善了处理器的管理。

PCI-X目前分为66MHz、100MHz和133MHz三个版本。工作于66MHz的PCI-X控制器将能访问最多4个PCI-X设备，当然，如果增加PCI-X至PCI-X的桥接芯片，那么可以支持更多的设备。66MHz PCI-X拥有533MB/s的带宽。PCI-X总线是共用的，有66MHz、100MHz和133MHz三种，100MHz PCI-X的设备均工作于100MHz下，此时PCI-X总线只能管理最多两个PCI-X设备，在64b总线和100MHz频率下，拥有800MB/s的带宽。最豪华的133MHz PCI-X工作于133MHz，将能提供惊人的1066MB/s带宽。

6.3.5　PCI-Express 总线

PCI-Express(简称PCI-E)是最新的总线和接口标准，它原来的名称3GIO，是由英特尔提出的。英特尔的意思是它代表着下一代I/O接口标准。交由PCI-SIG(PCI特殊兴趣组织)认证发布后才改名为PCI-Express。这个新标准将全面取代现行的PCI和AGP，最终实现总线标准的统一。它的主要优势就是数据传输速率高，目前最高可达到10GB/s以上。

PCI-Express带宽(双向传输模式)：

(1) 1 lane-x1：500MB/s。

(2) 4 lane-x4：2GB/s (2000MB/s)。

(3) 8 lane-x8：4GB/s (4000MB/s)。

(4) 16 lane-x16：8GB/s (8000MB/s)。

PCI-E总线是一种完全不同于过去PCI总线的一种全新总线规范，与PCI总线共享并行架构相比，PCI-Express总线是一种点对点串行连接的设备连接方式，点对点意味着每一个PCI Express设备都拥有自己独立的数据连接，各个设备之间并发的数据传输互不影响，而对于过去PCI那种共享总线方式，PCI总线上只能有一个设备进行通信，一旦PCI总线上

挂接的设备增多，每个设备的实际传输速率就会下降，性能得不到保证。现在，PCI-Express以点对点的方式处理通信，每个设备在要求传输数据的时候各自建立自己的传输通道，对于其他设备这个通道是封闭的，这样的操作保证了通道的专有性，避免其他设备的干扰。

6.3.6 现代微机总线配置

1. 南、北桥结构

现代微型计算机采用的就是多总线结构，如图6-12所示是其主板总线结构图。

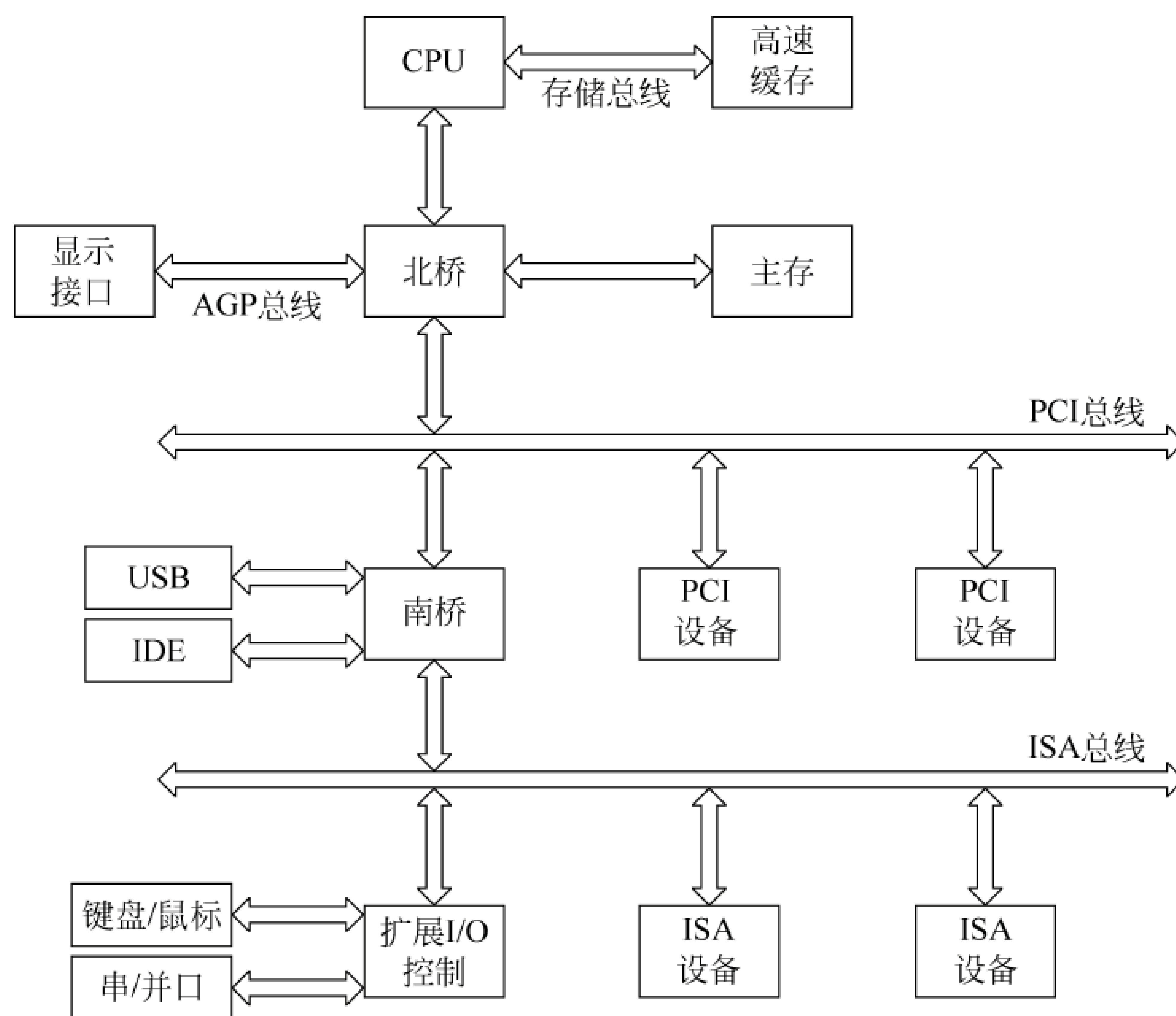


图6-12 现代微机主板总线结构示意图

其中，北桥芯片(North Bridge)是主板芯片组中起主导作用的最重要的组成部分，也称为主桥(Host Bridge)。一般来说，芯片组的名称就是以北桥芯片的名称来命名的，例如Intel 845E芯片组的北桥芯片是82845E，Intel 875P芯片组的北桥芯片是82875P，等等。北桥芯片负责与CPU的联系并控制内存、AGP数据等在北桥内部传输，提供对CPU的类型和主频、系统的前端总线频率、内存的类型(SDRAM，DDR SDRAM以及RDRAM等)和最大容量、AGP插槽、ECC纠错等的支持，整合型芯片组的北桥芯片还集成了显示核心。北桥芯片就是主板上离CPU最近的芯片，这主要是考虑到北桥芯片与处理器之间的通信最密切，为了提高通信性能而缩短传输距离。因为北桥芯片的主要功能之一是控制内存，而内存标准与处理器一样变化比较频繁，所以不同芯片组中北桥芯片是肯定不同的，当然这并不是说所采用的内存技术就完全不一样，而是不同的芯片组北桥芯片间可能存在一定的差异。

南桥芯片(South Bridge)是主板芯片组的另一个重要的组成部分，一般位于主板上离CPU插槽较远的下方，PCI插槽的附近，这种布局是考虑到它所连接的I/O总线较多，离处理器远一点有利于布线。南桥芯片不与处理器直接相连，而是通过一定的方式(不同厂商各

种芯片组有所不同)与北桥芯片相连。南桥芯片负责 I/O 总线之间的通信,如 PCI 总线、USB、LAN、ATA、SATA、音频控制器、键盘控制器、实时时钟控制器、高级电源管理等,这些技术一般相对来说比较稳定,所以不同芯片组中可能南桥芯片是一样的,不同的只是北桥芯片。所以现在主板芯片组中北桥芯片的数量要远远多于南桥芯片。例如早期英特尔不同架构的芯片组 Socket 7 的 430TX 和 Slot 1 的 440LX 其南桥芯片都采用 82317AB,而近两年的芯片组 845E/845G/845GE/845PE 等配置都采用 ICH4 南桥芯片,但也能搭配 ICH2 南桥芯片。更有甚者,有些主板厂家生产的少数产品采用的南北桥是不同芯片组公司的产品。南桥芯片的发展方向主要是集成更多的功能,如网卡、RAID、IEEE 1394 及 Wi-Fi 无线网络等。

传统的南北桥结构是通过 PCI 总线来连接的,常用的 PCI 总线是 33.3MHz 工作频率,32b 传输位宽,所以理论最高数据传输率仅为 133MB/s。由于 PCI 总线的共享性,当子系统及其他周边设备传输速率不断提高以后,主板南北桥之间偏低的数据传输率就逐渐成为影响系统整体性能发挥的瓶颈。因此,从英特尔 i810 开始,芯片组厂商都开始寻求一种能够提高南北桥连接带宽的解决方案。

2. 加速中心结构

加速中心结构(Accelerated Hub Architecture,AHA)首次出现在英特尔的著名整合芯片组 i810 中。在 i810 芯片组中,英特尔一改过去经典的南北桥架构,采用了新的 AHA 结构。AHA 结构由相当于传统北桥芯片的图形/存储器控制中心(Graphics & Memory Controller Hub,GMCH)和相当于传统南桥芯片的 I/O 控制中心(I/O Controller Hub,ICH),以及新增的固件控制器(Firmware Hub,FWH),它相当于传统体系结构中的 BIOS ROM,共三块芯片构成。

在这种新的 AHA 结构中,两块芯片不是通过 PCI 总线进行连接,而是利用能提供两倍于 PCI 总线带宽的专用总线连接的。这样,每种设备包括 PCI 总线都可以与 CPU 直接通信,Intel 810 芯片组中的内存控制器和图形控制器也可以使用一条 8b 的 133MHz“2×模式”总线,使得数据带宽达到 266MB/s,它的后续芯片组也大多采用 AHA 结构。

这种体系本质上跟南北桥结构相差不大,它主要是把 PCI 控制部分从北桥中剥离出来(北桥成为 GMCH),由 ICH 负责 PCI 以及其他以前南桥负责的功能,而 ICH 也采用了加速中心结构,在图形卡和内存与整合的 AC'97 控制器、IDE 控制器、双 USB 端口和 PCI 附加卡之间建立一个直接的连接。由于 AHA 结构提供了每秒 266MB 的 PCI 带宽,这使得 I/O 控制器和内存控制器之间可以传输更多的信息;再加上优化了仲裁规则,系统可以同时执行更多的线程,从而有了较为明显的性能提升。在 GMCH 与 ICH 之间的传输速率则达到了 8 位 133MHz DDR(等效于 266MHz,266MB/s),它使得 PCI 总线、USB 总线以及 IDE 通道与系统内存和处理器之间的带宽有较大的提高。

知识拓展

前端总线

前端总线的英文名称是 Front Side Bus,通常用 FSB 表示,这个名称是由 AMD 公司在推出 K7 CPU 时提出的,它指的是将 CPU 连接到北桥芯片的总线。北桥芯片负责联系内

存、显卡等数据吞吐量最大的部件，并和南桥芯片连接。CPU就是通过前端总线(FSB)连接到北桥芯片，进而通过北桥芯片和内存、显卡交换数据。

前端总线是CPU和外界交换数据的主要通道，因此前端总线的数据传输能力对计算机整体性能影响很大，如果没有足够快的前端总线，再强的CPU也不能明显提高计算机的整体速度。数据传输的最大带宽取决于所有同时传输的数据的宽度和传输频率，即数据带宽=(总线频率×数据位宽)/8。目前PC上所能达到的前端总线频率有266MHz、333MHz、400MHz、533MHz、800MHz等几种，前端总线频率越大，代表着CPU与北桥芯片之间的数据传输能力越大，更能充分发挥出CPU的功能。在同等条件下，前端总线越快，系统性能越好。

前端总线的频率与计算机外频是两个不同的概念，通常所说的外频指的是CPU与主板连接的速度，这个概念是建立在数字脉冲信号振荡速度基础之上的，而前端总线的速度指的是数据传输的速度，也就是说，100MHz外频特指数字脉冲信号在每秒钟振荡一亿次；而100MHz前端总线指的是每秒钟CPU可接受的数据传输量是100MHz×64b/8B/b=800MB/s。

6.4 外部总线接口

计算机与外部设备之间除了通过设备适配器经系统总线互连外，还可通过一些标准的外部总线连接。一般来讲，计算机主板上集成了一些常用的标准外部总线接口，如USB接口、IEEE 1394接口、SCSI等。

6.4.1 SCSI

SCSI是小型计算机系统接口(Small Computer System Interface)的缩写，是一种外部总线接口标准。SCSI是由美国的Shugart Associates和NCR公司在1979年发明的。Shugart Associates是当时磁盘驱动器的主要制造厂商(希捷公司前身)，而NCR也曾经在小型计算机市场中扮演重要的角色。这个接口开始的名字为SASI，即Shugart Associates Systems Interface，1982年4月美国国家标准委员会ANSI将其改名为SCSI，使其有一个更通用的名字。1986年6月ANSI发布了SCSI-1标准，从此真正开始了SCSI的应用和发展历程。

SCSI正如其名字所表示的那样，最初用于小型计算机系统，随着微型计算机技术的发展，后来也用于微机系统，但主要应用于对性能要求较高或专业性较强的场合，如网络服务器等。

1. SCSI的特点

与微型计算机中的IDE接口相比，SCSI具有以下几个方面的特点。

1) 应用面广

SCSI是一种连接主机和外围设备的接口，它不仅仅用于连接磁盘驱动器、磁带机、光驱等辅存设备，还支持其他多种输入输出设备。

2）适应性强

SCSI接口适配器（又称SCSI控制器）中包含了一个相当于小型CPU功能的控制逻辑，它有自己的命令集和缓冲存储器。SCSI的命令与设备和主机无关，采用命令描述块（CDB）的形式由主设备发送给目标设备，CDB说明了操作的性质、源和目的数据块的地址、传送的块数等信息。无论主机采用什么类型的设备级接口、设备甚至系统总线结构，SCSI总线都有相同的物理和逻辑特性。另外SCSI接口控制器能够独立完成输入输出过程中的数据缓冲、数据格式转换及DMA控制等，从而可以减轻CPU的负担，有效提高CPU的利用率。

3）扩展性好

SCSI系统可以是一个主机，即一个主适配器和一个外设控制器最简单的形式，也可以是一个或多个主机与多个外设控制器的组成。SCSI规定系统至多有的SCSI设备数目为SCSI总线数据的位数，如采用32位数据总线，则至多有32个SCSI设备。而微型计算机上的IDE接口最多只能连接4个设备。因此，SCSI具有良好的可扩展性。

4）速度快

SCSI总线在刚推出时最大的数据传输速率仅为5MB/s，而新一代的SCSI标准最大的数据传输率可达640MB/s，甚至上千MB/s。

但SCSI与IDE在价格方面，其价位高出许多，而且在使用方面不如IDE方便。目前，SCSI的设计正朝着支持即插即用、多功能化及网络化的方向发展。

2. SCSI家族

SCSI有多种版本，但作为标准来说，主要包括三种：SCSI-1、SCSI-2和SCSI-3，另外即将推出的SCSI-4和SCSI-5也将加入SCSI家族的行列。

SCSI家族类型及各自性能特征如下。

（1）SCSI-1：它是最早的SCSI，在1979年由Shugart制订，并于1986年由ANSI发布正式标准。它支持8位并行数据传输，可同时连接7台SCSI设备，最大数据传输率为5MB/s。

（2）SCSI-2：它是继SCSI-1后的接口标准，于1992年推出，也称为Fast SCSI（快速SCSI）。其中，采用8位并行数据传输的SCSI称为Fast SCSI（Narrow模式），它的数据传输率为10MB/s，最大支持连接设备数为7台；后来出现的采用16位并行数据传输的SCSI称为Fast Wide SCSI（快速宽带SCSI），它的数据传输率提高到了20MB/s，最大支持连接设备数为15台。

（3）SCSI-3：它是在SCSI-2之后推出的Ultra SCSI（超级SCSI）类型，在这个大类中也按数据位宽的不同先后推出了两个小类。采用8位并行数据传输时称为Ultra SCSI，它的数据传输率为20MB/s，最大支持连接设备数为8台。在将并行数据传输的总线带宽提高到16位后出现了Ultra Wide SCSI，它的传输率又成倍提高，达到了40MB/s，最大支持连接设备数为15台。

（4）Ultra2 SCSI：它是在Ultra SCSI的基础上推出的SCSI接口类型。于1997年提出，采用了低电平微分（Low Voltage Differential，LVD）的传输模式，允许接口电缆最长为12m，这大大增加了设备的灵活性。与上面几种SCSI一样，它也分为采用8位的Narrow模式和采用16位的Wide模式。8位的Narrow模式即为Ultra2 SCSI，它的传输率为

40MB/s，最大支持连接设备数为7台；而采用16位的Wide模式则称为Ultra2 Wide SCSI，它将传输率提高到了80MB/s，最大支持连接设备数为15台。

（5）Ultra3 SCSI：它是Ultra2 SCSI的更新接口，于1998年9月份提出，它除支持现有的SCSI规格，使用和Ultra2 SCSI完全一样的接口电缆及终结器外，还包含了一些新功能。首先Ultra3 SCSI采用双缘传输频率（Double Transition Clocking），而Ultra2 SCSI采用的是单缘传输频率，因此Ultra3 SCSI的传输率是前者的两倍，即最高可达160MB/s；此外Ultra3 SCSI还提供了领域确认、CRC冗余循环校验、封包协议、快速仲裁选取等几项新功能；为了加快Ultra3 SCSI新技术的推出，很多厂商首先推出了Ultra160 SCSI。Ultra160 SCSI的技术和Ultra3 SCSI一样，只是没有快速仲裁选取和封包协议这两项功能，可以说Ultra160 SCSI就是Ultra3 SCSI的子集。

（6）Ultra320 SCSI：它的全称为Ultra320 SCSI SPI-4技术规范。Ultra320 SCSI单通道的数据传输速率最大可达320MB/s，如果采用双通道SCSI控制器可以达到640MB/s。从基础架构的发展来看，160MB/s到320MB/s的提升在技术上并不复杂，花费也不大，因此对于系统集成商来说，服务器从SCSI Ultra160 SCSI到Ultra320 SCSI的技术过渡是非常容易实现的。

3. SCSI的配置结构

一个SCSI适配器通过一组SCSI总线与多个SCSI设备连接，它们构成一个独立的I/O子系统，称为一个SCSI域。在一台机器中可以有多个SCSI域，如图6-13所示。其中ID编号用来标识一个域中的不同设备，不同域的ID编号之间没有关系，可以重复。

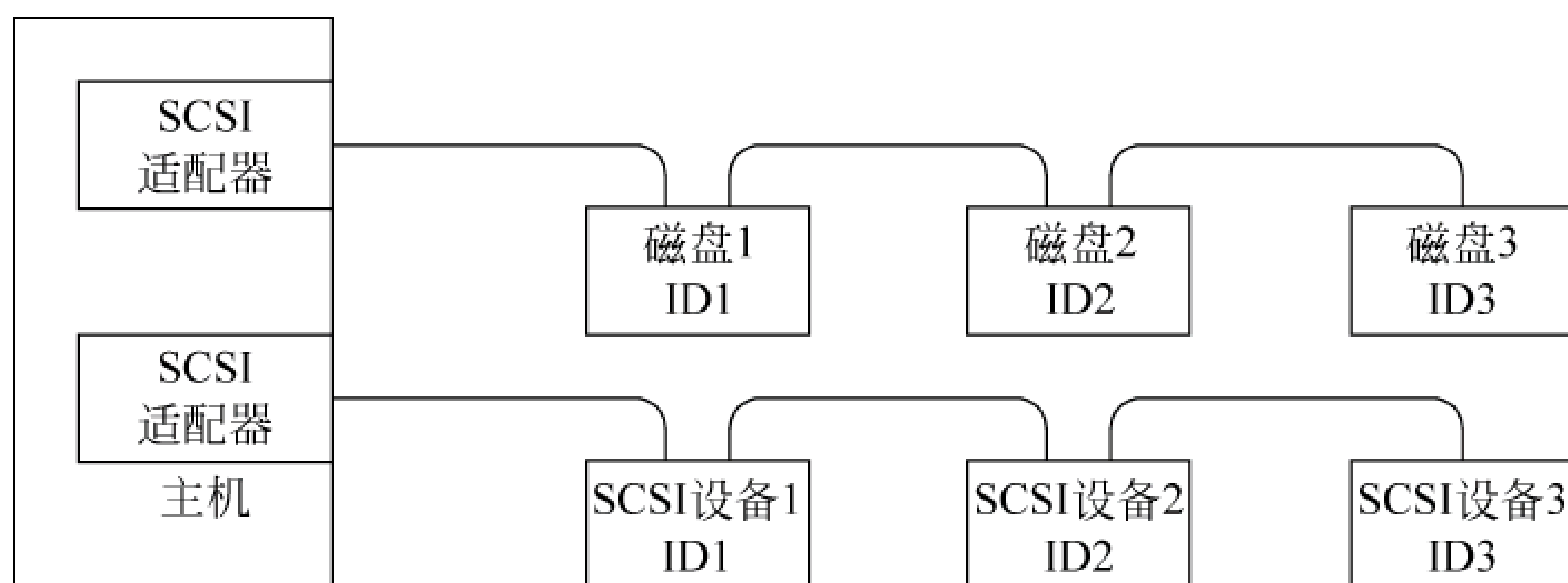


图6-13 SCSI的配置

SCSI设备是沿着一条SCSI总线构成菊花链接的结构，即一个SCSI设备的输出利用SCSI电缆连接到另一个设备的输入。CPU只与其SCSI适配器进行通信，当有需要时发布各种I/O命令，在接下来的时间里，CPU执行自己的任务，而由SCSI适配器来负责管理输入输出操作。

4. SCSI的接口类型

SCSI连接器分为内置和外置两种，内置接口的外形和IDE接口类似，只是针数和规格稍有差别，主要用于连接光驱和硬盘。外置接口主要包括50针接口、68针接口和80针接口等，如图6-14所示。

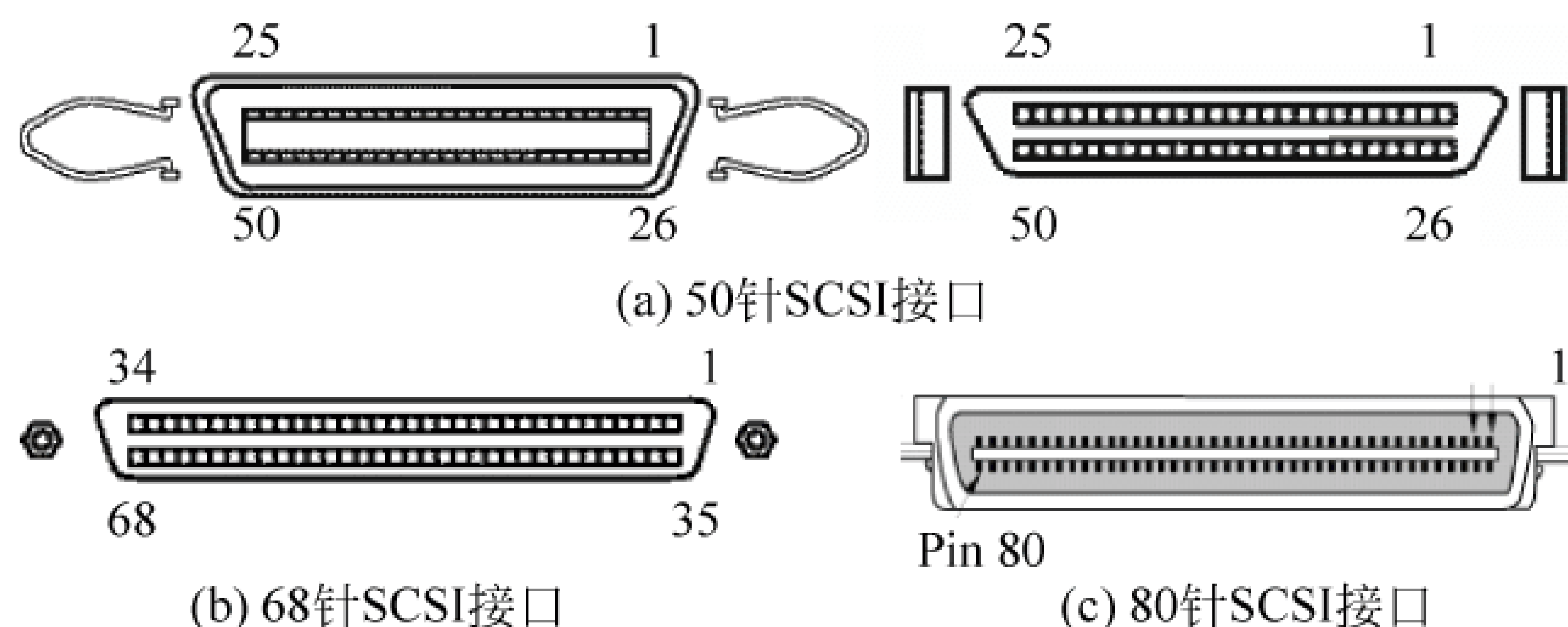


图 6-14　SCSI 接口的类型

1）DB-50 连接器

连线上使用 50 针公接头，人们通常称做“大 50 接头”（Centronics 50-pin）。DB-50 的使用率较高，许多大型的外接设备都采用此种接头。但它的体型庞大，不适合应用在接口卡上。它的传输率不高，应用在 SCSI-1、Fast SCSI 等级的设备上。

2）HD-50 连接器

连线上使用 50 针公接头 HD-50（High Density 50-pin），人们通常称做“小 50 接头”，还有 Micro DB-50、Mini DB-50 等名称，除了可接 Fast SCSI 等级设备之外，还可用于连接 Ultra SCSI 等级的高速设备。由于接头体型小，同样也是 50 针高密接头（比较容易转接）。它几乎成为了 SCSI 卡的标准外接接头。HD-50 的传输率比 DB-50 快。

3）HD-68 连接器

连线上使用 68 针公接头 HD-68（High Density 68-pin），和 HD-50 同属于高密接头，但针脚比 HD-50 多，适用于 Wide、Ultra2 SCSI 等级的高速设备，使用较少。

4）HD-80 连接器

在 Ultra SCSI 等级的高速设备上使用，使用较少。

6.4.2　IEEE 1394 接口

随着计算机技术与家电技术的融合，多媒体信息进入家庭只不过是个时间的早晚问题，多媒体信息传输自然就成为家电厂商与计算机厂商所共同关心的问题。多媒体数据传输首先要求的是实时性，现有的单纯图形传输或网络应用一般对实时性要求并不高。例如在 Internet 上浏览主页，虽有延迟，但一般尚可接受；如果是在网络上举行会议，实时性就会变成一个要求十分苛刻的问题。然后是连接的方便性，只有高性能但使用不方便同样难以进入家用产品市场，家电和计算机融合就无从谈起。新的接口必须使用简便才不会影响其推广应用。第三是通用性，只有不局限于某种特定的环境和设备，才会拥有广泛的应用群体。最后是价格便宜，价格太贵，用户数量就会受到限制。

IEEE 1394 正是在这样一种背景下被提出来的。它最初是由美国 Apple 苹果电脑公司在 20 世纪 80 年代中期开始研发。当初苹果公司称之为 FireWire（火线），而之后的索尼公司则称之为 i. Link，德州仪器公司称之为 Lynx。苹果公司的 FireWire 标准于 1987 年制定完成，但直到 1994 年 9 月才由 IEEE 成立了 IEEE 1394 行业协会，由 Adaptec、AMD、Apple、Cirrus Logic、IBM、Microsoft、Molex、Philips、Skip stone、Sony 和 TI 等公司组成执行委员会，制定了总线接口标准，即 IEEE 1394-1995 技术规范。

IEEE 1394 是一个高速串行总线接口标准，既可作为总线标准应用于计算机主板，也可作为外部接口标准应用于计算机与各种外部设备的连接，尤其是与一些现代数码电器产品的连接，如实现与数码相机、数码摄像机及各种数字音频视频设备之间的高速、宽带的数据传输等。

1. IEEE 1394 接口的特点

IEEE 1394 的特点可以归纳为以下几个方面。

1) 高速率

IEEE 1394-1995 中规定标准数据传输速率为 100～400Mb/s。新的 IEEE 1394b 中更高的速度是 800Mb/s～3.2Gb/s。其实 400Mb/s 就几乎可以满足所有的要求。实际上，数字音视频数据一般都要经过压缩再进行传输，因此，一般来讲 200Mb/s 已经能够满足实际音视频数据传输需要的速度了。但对多路数字音视频信号传输来说，传输速率总是越高越好、永无止境。

2) 传输距离长

虽然 IEEE 1394-1995 标准允许其总线长度只有 4.5m，但新的 IEEE 1394b 标准可以实现 100m 范围内的设备互连，这样 IEEE 1394 设备的应用就更加方便了。

3) 支持热插拔和即插即用

IEEE 1394 支持带电插拔设备，接插或拔出装置不必关闭电源。同时它还支持即插即用，增加新装置无须设置，系统可自动识别并配置，只要设备支持 IEEE 1394 这一标准即可。现在主流的 Windows 操作系统都对 IEEE 1394 有很好的支持，在这些操作系统中用户不用再安装驱动程序，也能使用 IEEE 1394 设备。

4) 接口简单、廉价

IEEE 1394-1995 标准规范规定总线中包含 4 根信号线和 2 根电源线，使用细缆连接，这使得连接和安装都十分简单。IEEE 1394 的控制软件和连接导线的实现成本都比并行总线要低，而且不需要解决信号干扰问题，因此成本更加低廉。

5) 互连设备多

虽然 IEEE 1394 最多只可支持 63 个节点的 1394 设备串连，但可以满足绝大多数应用的需要。

6) 支持对等传输

IEEE 1394 对对等传输的支持也是非常吸引人的，这样两台 IEEE 1394 设备无须通过计算机即可实现点到点的直接相连和数据传输。这意味着只要设备支持，就可以方便地将如数码相机等与具有 IEEE 1394 接口的硬盘等连接，并直接将数码相机中的数据存储到硬盘中。

7) 支持同步和异步传输

IEEE 1394 同时支持同步和异步两种数据传输方式。在异步传输方式下，信息的传输可以被中断；而在同步传输方式下，数据将在不受任何中断和干扰的情况下实现连续的传输。同步方式可以很好地应用于实时视频数据流的传输，满足视频画面质量的要求。

8) 灵活的传输模式

IEEE 1394 的传输模式主要有 Backplane 和 Cable 两种。其中 Backplane 模式是一种

基于主板的总线模式,其所实现的数据传输速率分别为12.5MB/s、25MB/s、50MB/s,可以用于多数带宽要求不是很高的应用环境,如Modem(包括ADSL、Cable Modem)、打印机、扫描仪等。而Cable模式是一种快速接口模式,其所能达到的数据传输速率分别为100MB/s、200MB/s和400MB/s几种,主要应用于一些数码设备的数据传输。

总之,IEEE 1394既是新一代接口,又是新一代总线;既是计算机外设接口标准,又是家电接口标准,目前已广泛应用于家庭和办公等诸多领域。

2. IEEE 1394的配置结构

IEEE 1394采用菊花链式配置,也允许树状结构配置。事实上,菊花链结构是树状结构的一种特殊情况。

IEEE 1394接口也需要一个主适配器和系统总线相连。在PC中,这个主适配器的功能逻辑集成在主板的核心芯片组的桥芯片中,称之为主端口。主端口是1394接口树状配置结构的根节点。一个主端口最多可连接63台设备,这些设备称为节点,它们构成亲子关系。如图6-15所示给出了一个IEEE 1394配置结构图。

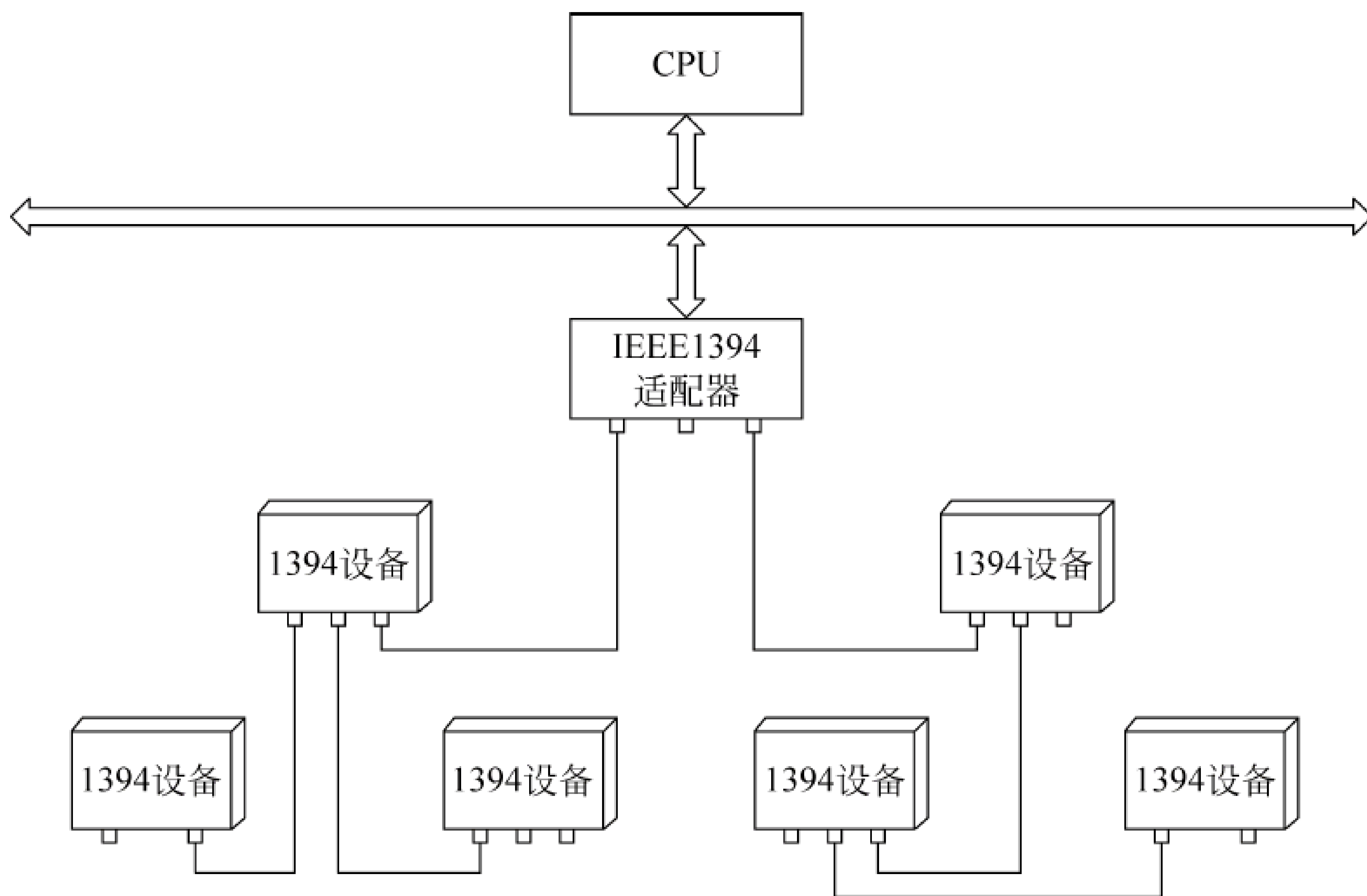


图6-15 IEEE 1394配置结构图

IEEE 1394采用集中式总线仲裁方式。中央仲裁逻辑在主端口内,并以先到先服务的方法来处理节点提出的总线访问请求。在n个节点同时提出使用总线请求时,按照优先权进行仲裁。最靠近根节点的竞争节点有高的优先权;同样靠近根节点的竞争节点,其设备标识号ID大的有更高的优先权。

为了保证总线设备的对等性和数据传输的实时性,IEEE 1394的总线仲裁增加了均等仲裁和紧急仲裁功能。均等仲裁是将总线时间分成均等的间隔,当间隔期间开始时,竞争的每个节点置位自己的仲裁允许标志,在间隔期内各节点可竞争总线的使用权。一旦某节点获得总线访问权,则它的仲裁允许标志被复位,在此期间它不能再去竞争总线,以此来防止具有高优先权的忙设备独占总线。紧急仲裁指对某些高优先权的节点可为其指派紧急优先权,具有紧急优先权的节点可在一个间隔期内多次获得总线控制权,允许它控制75%的总

线可用时间。

3. IEEE 1394 的接口类型

IEEE 1394 有 6 针和 4 针两种接口类型，也就是常说的大口和小口，如图 6-16 所示。6 针接口外观为六角形，4 针接口外观为小型四角形。最早苹果公司开发的 IEEE 1394 接口是 6 针的，后来，SONY 公司看中了它数据传输速率快的特点，将早期的 6 针接口进行改良，重新设计成为现在大家所常见的 4 针接口，并且命名为 i. Link。这种连接器如果要与标准的 6 导线线缆连接的话，需要使用转换器。

图 6-16　IEEE 1394 接口

6 针和 4 针两种接口的区别在于能否通过连接线缆向所连接的设备供电。6 针接口中有 4 针是用于传输数据的信号线，另外 2 针是向所连接的设备供电的电源线。对于低电源设备，电缆中提供的电源可以满足设备的电源需求，因而无须配备外接电源连接器，这就是 6 针接口的优点。而 4 针接口则省去了 2 根电源线，剩下的 4 根线全部为数据信号线。由于接口电缆未含电源线，因此，使用 4 针接口连接的设备必须单独供电。

在应用方面，一般来讲，6 针的接口主要用于普通的台式计算机，很多主板都整合了这种接口；在笔记本和一体机等计算机中则大多采用 4 针。另外，在数码和家电产品中，采用 4 针的情况也比较多。特别是近一段时间，笔记本电脑和 DV 都在朝着小型化和超薄化方向发展，4 针接口在外观尺寸上要比 6 针的小很多，因此占用空间小。另外，并非所有设备都需要通过接口供电，例如很多 DV 的 1394 接口主要用于传输影像数据，所以无须在接口上供电。而还有些设备仅靠接口电缆上的供电满足不了要求，例如像硬盘这种对于电量要求较高的设备就很难从其所接入的设备中得到充足的电力供应，在一定场合下需要使用一个外接电源。

6.4.3　USB 接口

人们对 PC 机箱背面的几个标准外设接口应该已经很熟悉了，一是串行接口（又称 COM 口），二是并行接口，三是 PS2 口。串行接口支持的是串行数据传输，其数据传输速度可以达到 115～230kb/s，一般被用来连接早期的鼠标和 Modem 等慢速设备；并行接口支持的是并行数据传输，其数据传输率能达到 1MB/s 左右，一般被用来连接打印机、扫描仪等对速率要求较高的外设；PS2 口则用来连接现代 PC 的键盘和鼠标。

对串行接口和并行接口而言，一方面它们所能达到的数据传输速度低；另一方面每个串口或并口上只能连接一个设备，所以，它们都具有扩展能力差的先天不足。另外，虽然现在很多串行/并行设备都可以实现即插即用，但热插拔仍然是个危险的动作。

而 USB 正是为了解决速度、扩展能力和易用性这三个问题而提出的低成本解决方案。1994 年，Intel、Compaq、Digital、IBM、Microsoft、NEC、Nortel 7 家计算机和通信厂商为了解决上述问题，联合成立了 USB 论坛，并分别于 1996 年、1998 年、1999 年、2008 年和 2013 年正式制订了 USB 1.0、USB 1.1、USB 2.0、USB 3.0 及 USB 3.1 规范，用于形成统一的 PC 外设接口标准。

USB论坛于1996年1月正式提出USB 1.0规格，带宽为12MB/s。不过因为当时支持USB的周边装置少得可怜，所以主机板厂商不太把USB口直接设计在主机板上。

USB 1.1规范在1998年9月提出用来修正USB 1.0，但传输的带宽不变，仍为12MB/s。USB 1.1向下兼容于USB 1.0，

USB 2.0技术规范是在1999年发布的，把外设数据传输速度提高到了480MB/s，是USB 1.1设备的40倍。

USB 3.0标准在2008年11月18日公开发布，新规范提供了十倍于USB 2.0的传输速度(5Gb/s)和更高的节能效率，可广泛用于PC外围设备和消费电子产品。

USB 3.1规范在2013年发布。新标准在接口方面没有什么改变，但它可以提供两倍于USB 3.0的传输速度(即10Gb/s)，同时还能向下兼容USB 2.0。

目前，USB接口已成为PC上标准配置的接口类型，其应用也越来越广泛，除了一些计算机传统的设备如键盘、鼠标、打印机等逐渐采用USB接口方式与主机连接外，各种家电产品也纷纷加入了USB的行列。

1. USB接口的特点

USB的全称是Universal Serial Bus，即通用串行总线，其主要特点如下。

1) 高速度

与传统的串行接口和并行接口相比较，USB接口提供的数据传输速度更快。其中USB 1.1提供全速(12MB/s)和低速(1.5MB/s)两种速率来适应各种不同类型的外设，而USB 2.0的传输速度可以达到480MB/s，USB 3.0的传输速度可以高达5Gb/s，目前已发展到USB 3.1版本，数据传输速率达10Gb/s。

2) 即插即用和支持热插拔

与IEEE 1394接口一样，系统对USB设备能够自动识别和自动配置，并且允许在不关闭电源的情况下对USB设备直接插拔。

3) 可以连接多个设备

IEEE 1394最多只可支持63个节点的1394设备连接，而USB所连接的设备更多。USB设备可以直接连在PC上任意的USB接口，还可以使用USB Hub进行扩展，使更多的USB设备连接到系统中。USB的Hub有一个上行端口用于连接到Host，多个下行端口连接其他的设备，它们以一种分层星型结构的方式互连，使整个系统最多可以扩展连接127个设备，这足以满足各种应用的需要。

4) 支持多种数据传输方式

针对设备对系统资源需求的不同，USB规范规定了4种不同的数据传输方式：一是控制传输方式，这是一种双向传送方式，数据量通常比较小。二是同步传输方式传送，这种传输方式提供了确定的带宽和间隔时间，它被用于时间严格并具有较强容错性的流数据传输，或者用于要求恒定的数据传送率的即时应用中。三是中断传输方式，这种传输方式典型的应用在少量的、分散的、不可预测数据的传输，键盘、操纵杆和鼠标就属于这一类型。四是大量数据传输方式，主要应用在大量数据的传输和接收上，打印机和扫描仪等属于这种类型。

2. USB的配置结构

USB的物理连接采用的是一种分层的星型结构,USB Hub是每个星型结构的中心,主机则相当于根Hub,所有USB设备或者直接与主机相连,或者与USB Hub相连。USB Hub可以级联,最多可达5层,所连接设备最多为127个。与IEEE 1394稍有不同的是,USB设备只能上连到主机或USB Hub,设备之间不能互连;而IEEE 1394的设备则可以互连成菊花链结构以及对等互连。对于这一点,通过如图6-17所示的USB配置结构和如图6-15所示的IEEE 1394配置结构的比较就可以很容易地理解。

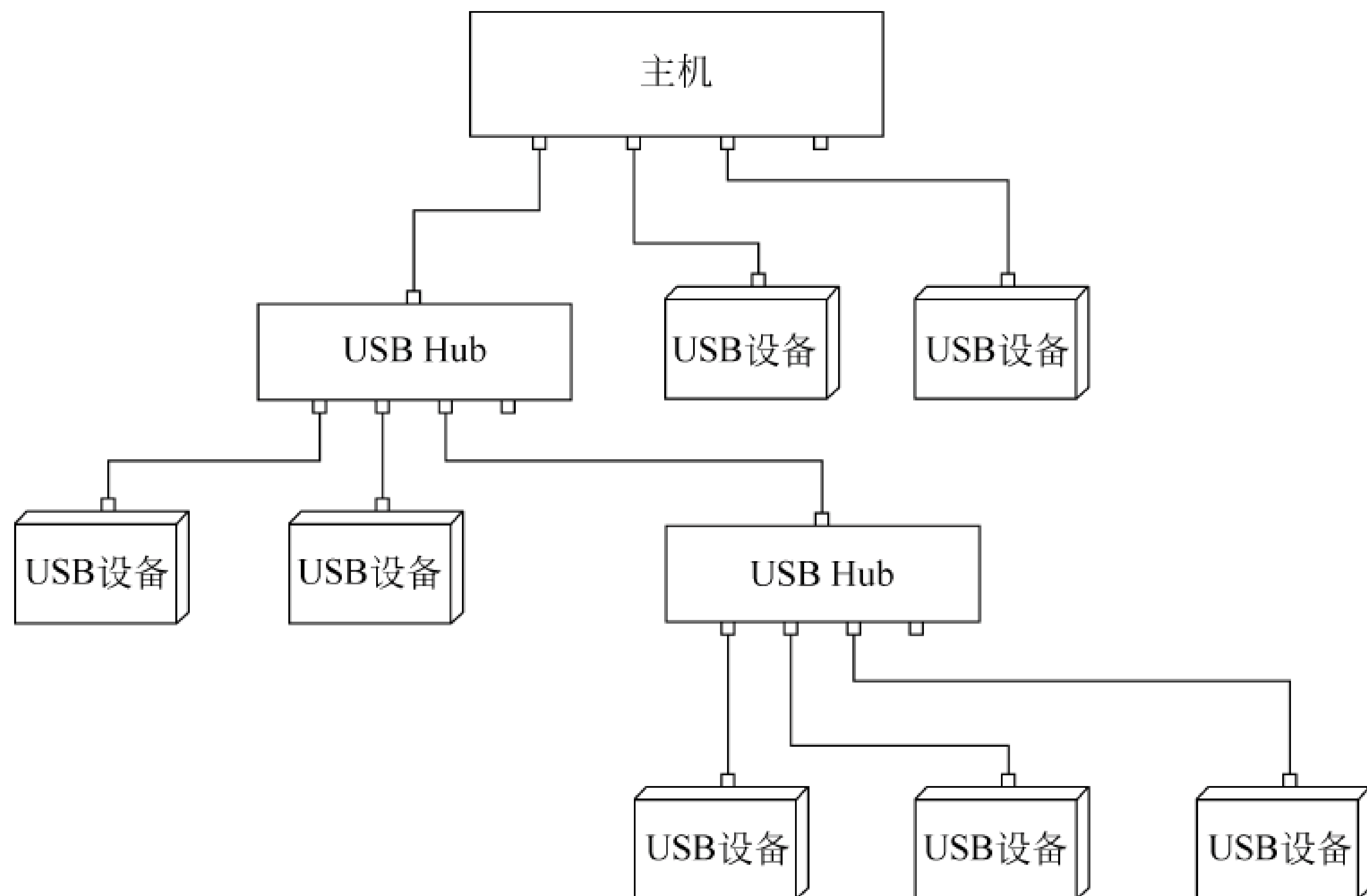


图6-17 USB配置结构

3. USB的接口类型

所有USB设备都需要通过USB电缆来实现与计算机主机的物理连接才能正常使用。USB电缆分为屏蔽型和非屏蔽型两种,人们使用的大多数都是非屏蔽型的USB电缆。不管何种类型的电缆,它们的接头都是一样的,一端为扁形的接口,称为上行端口,用于连接主机或USB Hub的扩展端口;另一端为方形接口,称为下行端口,用于连接USB设备及USB Hub,如图6-18所示。

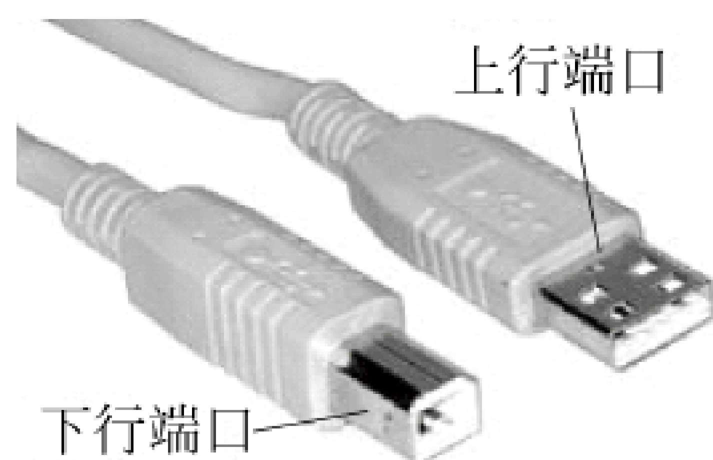


图6-18 USB接口

USB通过一根4线电缆来传输信号和电源,其中两线组成一对传输数据的差模信号线,另两线则提供了+5V的电源,可以有条件地给一些设备供电。

USB设备的供电通常有两种方式。一种是设备本身不带独立的电源适配器,电源由主机的USB接口直接供应。这种设备大多是耗电量不大的纯电路类产品,如USB接口的Modem、鼠标、U盘、MP3等;另一种USB设备需要单独供电,这类设备通常都是机电一体的装置,如USB接口的扫描仪、打印机等,因为这些设备的机械传动部分需要大功率的电源才能驱动。

知识拓展

串行传输还是并行传输?

从技术发展的情况来看,USB 取代 IEEE 1284,SATA 取代 PATA,PCI Express 取代 PCI……似乎串行传输方式大有彻底取代并行传输方式的势头。

从原理来看,并行传输方式其实优于串行传输方式。那么,为何现在的串行传输方式会更胜一筹?下面将从并行、串行的变革以及技术特点分析隐藏在其表面现象背后的深层原因。

计算机中的总线和接口是主机与外部设备间传输数据的"大动脉",随着处理器速度的节节攀升,总线和接口的数据传输速度也需要逐步提高,否则就会成为计算机发展的瓶颈,从计算机中使用的 ISA 总线发展到 PCI 总线就足以说明这一点。

并行数据传输技术向来是提高数据传输率的重要手段,但是,进一步发展却遇到了障碍。首先,由于并行传输方式的前提是用同一时序传播信号,用同一时序接收信号,而过分提升时钟频率将难以让数据传输的时序与时钟合拍,布线长度稍有差异,数据就会以与时钟不同的时序送达。此外,提升时钟频率还容易引起信号线间的相互干扰,因此,并行方式难以实现高速化。另外,增加位宽无疑会导致主板和扩充板上的布线数目随之增加,成本随之攀升。

至于如果有人问关于串行传输与并行传输谁更好?目前只能说"在相同频率下并行传输速率更高"这个基本道理是永远不会错的,通过增加位宽来提高数据传输率的并行策略仍将发挥重要作用。当然,前提是有更好的措施来解决并行传输的种种问题。串行传输现在之所以走红,是由于将单端信号传输转变为差分信号传输,并提升了控制器工作频率的原因。

技术进步周而复始,以至无穷,没有一项技术能够永远适用。计算机技术将来跨入 THz 时代后,对信号传输速度的要求会更高,差分传输技术是否能满足要求?是否需要另一种更好的技术来完成频率的另一次突破呢?不妨拭目以待!

本章小结

本章主要讲述了以下内容:

(1) 总线概述。讲述了总线的基本概念、总线的类型、总线的规范和单总线、局部总线、多总线结构等,介绍了总线的控制方式,包括总线的仲裁、总线的定时和总线的数据传输模式等。

(2) 总线标准及总线举例。介绍了几种应用非常广泛的系统总线(以 ISA 总线为例)和局部总线(PCI 总线、PCI-X 总线以及 PCI-Express 总线),介绍了这些总线的特点、配置结构及应用。

(3) 外部总线接口标准。讲述了小型计算机系统接口 SCSI 和两种新型的高速串行总线 IEEE 1394 和 USB,这两种总线接口标准在计算机中得到了越来越广泛的应用,尤其是对于今后计算机与家用电器的连接和融合将起到越来越重要的作用。

习题六

一、名词解释

1. 总线　2. 局部总线　3. 外部总线　4. 前端总线

二、选择题

1. 下列有关对总线的描述不正确的是(　　)。
   A. 总线是可共享的　　B. 总线是可独占的
   C. 总线的数据传输是串行的　　D. 通过总线仲裁实现对总线的占用
2. 通过总线可以(　　)。
   A. 减少部件之间的连接信号线　　B. 提高部件之间的传输速度
   C. 增加数据信号线的条数　　D. 增加地址信号线的条数
3. 系统总线是用于连接(　　)。
   A. 存储器各个模块　　B. CPU、存储器和I/O设备
   C. 主机与I/O设备　　D. 计算机与计算机
4. 下列描述PCI总线正确的是(　　)。
   A. PCI总线是一个与处理机无关的高速外围总线
   B. PCI总线的基本传输机制是串行传输
   C. PCI设备一定是主设备
   D. 系统中只允许有一条PCI总线
5. 下列不属于外部总线标准的是(　　)。
   A. IEEE 1394　　B. USB　　C. SCSI　　D. ISA
6. 一次总线事物中,主设备只需给出一个首地址,从设备就能从首地址开始的若干连续单元格读出或写入的个数,这种总线事务方式称为(　　)。(2014年全国硕士研究生入学统一考试计算机学科专业基础综合试题)
   A. 并行传输　　B. 串行传输　　C. 突发　　D. 同步
7. 下列选项中,用于设备和设备控制器(I/O接口)之间互连的接口标准是(　　)。(2013年全国硕士研究生入学统一考试计算机学科专业基础综合试题)
   A. PCI　　B. USB　　C. AGP　　D. PCI-Express
8. 某同步总线的时钟频率为100MHz,宽度为32位,地址/数据线复用,每传送一次地址或者数据占用一个时钟周期。若该总线支持突发(猝发)传输方式,则一次"主存写"总线事务传输128位数据所需要的时间至少是(　　)。(2012年全国硕士研究生入学统一考试计算机学科专业基础综合试题)
   A. 20ns　　B. 40ns　　C. 50ns　　D. 80ns
9. 下列关于USB总线特性的描述中,错误的是(　　)。(2012年全国硕士研究生入学统一考试计算机学科专业基础综合试题)
   A. 可实现外设的即插即用和热拔插
   B. 可通过级联方式连接多台外设
   C. 是一种通信总线,连接不同外设

D. 同时可传输 2 位数据,数据传输率高

10. 下列选项中,在 I/O 总线的数据线上传输的信息包括(　　)。(2012 年全国硕士研究生入学统一考试计算机学科专业基础综合试题)

Ⅰ. I/O 接口中的命令字　　Ⅱ. I/O 接口中的状态字　　Ⅲ. 中断类型号

A. 仅Ⅰ、Ⅱ　　B. 仅Ⅰ、Ⅲ　　C. 仅Ⅱ、Ⅲ　　D. Ⅰ、Ⅱ、Ⅲ

三、综合题

1. 为什么总线结构适合于计算机系统中各部件的互连?再列举几种书中没有提到的互连结构,并与总线结构进行比较。

2. 总线两个最主要的特点是什么?

3. 总线中都有哪些类型的信号线?分别用于传输哪种信息?

4. 总线按功能及所连接的部件种类可划分为哪几类?

5. 什么是系统总线?列举几种计算机中常用的系统总线。

6. 什么是总线带宽?

7. 总线仲裁主要实现什么功能?有哪些总线仲裁方式?

8. 按总线定时方式的不同,有哪两种总线数据传输方式?

9. 总线标准主要定义了哪几个方面的特性(规范)?

10. PCI 总线提供了哪几类“桥”?分别起到什么作用?

11. 试将 SCSI 与 IDE 接口进行对比。

12. 试将 IEEE 1394 接口与 USB 接口进行对比。

13. 设某数据总线为 32 位,总线时钟频率为 33MHz,求该数据总线的带宽。

14. 观察自己使用的家用电器产品,看看它们分别使用了什么样的接口标准,在与计算机的连接上是怎样使用和操作的。

第7章 CPU组织与结构

将运算器、控制器集成在一个芯片上称为中央处理器(CPU),随着超大规模集成电路(VLSI)技术的发展,CPU 中还集成了一级或多级高速缓冲存储器(cache)。CPU 是计算机系统的核心组成部件,在微型计算机中,又把 CPU 称为微处理器。

本章首先介绍 CPU 的功能与组成;然后介绍 CPU 的指令周期及执行指令的过程;最后介绍 CPU 控制部件设计的两种主要方法:硬布线设计法和微程序设计法。

7.1 CPU 的功能和组成

随着集成电路设计、制造及工艺水平的进步,芯片的集成度越来越高,可以制造出功能越来越强大的集成电路,这使得计算机的中央处理器(CPU)的性能越来越强。虽然不同时期、不同厂家生产的 CPU 在功能、组成结构上有许多差异,但 CPU 的基本功能——对机器指令的执行和数据的处理进行全过程的控制,不会改变。

7.1.1 CPU 的功能

第 3 章介绍了指令系统,指令系统是一台计算机所能够执行的全部指令集合。当用计算机解决某个问题时,首先必须为它编写程序或者利用已有的程序,程序是由一系列指令按解决问题的步骤构成的,这个序列明确地告诉计算机应该执行什么操作,在什么地方找到用来操作的数据。程序一旦装入主存储器,计算机就能按照程序规定的顺序自动、逐条地取出指令,对指令进行译码并执行指令,直到指令序列全部执行完毕,程序所要求的功能及对信息进行的加工和处理也就完成了。因此,整个计算机工作的过程就是不断地取出指令、对指令进行译码并执行指令的过程。专门用来完成此项工作的计算机部件称为**中央处理器**(Central Processing Unit,CPU)。

CPU 对于整个计算机系统的运行是极其重要的,归纳起来它具有以下 4 个方面的基本功能。

(1) 指令控制。

程序的顺序控制称为指令控制。由于程序是一个指令序列,这些指令的顺序不能任意颠倒,必须严格按程序规定的顺序进行。

(2) 操作控制。

一条指令的功能往往是由若干个操作信号的组合来实现的,因此,CPU 管理并产生由

内存(或指令 cache)取出的每条指令的操作信号,把各种操作信号送往相应的部件,从而控制这些部件按指令的要求进行动作。

(3) 时间控制。

对各种操作进行时间上的定时称为时间控制。在计算机中,各种指令的操作信号以及一条指令的整个执行过程都将受到时间的严格定时。

(4) 数据加工。

数据加工就是对数据进行算术运算和逻辑运算处理,并进行逻辑测试等。

在执行指令的过程中,如出现运算的溢出错误、存储器校验错及外部设备的服务请求等异常情况时,必须转入异常处理;异常处理一般由中断机构执行中断服务程序来完成。所以,取指令、分析指令并执行指令是 CPU 本质的功能。

7.1.2　CPU 的基本组成

在早期的计算机中,运算器和控制器是组成计算机的两个部件,后来出现了集成电路技术,把运算器和控制器集成在一个芯片中称为中央处理器(CPU)。

CPU 有通用 CPU 和嵌入式 CPU。通用和嵌入式的分别,主要是根据应用模式的不同而划分的。通用 CPU 芯片的功能一般比较强,能运行复杂的操作系统和大型应用软件;嵌入式 CPU 在功能和性能上有很大的变化范围。随着集成度的提高,在嵌入式应用中,人们倾向于把 CPU、存储器和一些外围电路集成到一个芯片上,构成所谓的系统芯片(简称为 SoC),而把 SoC 上的那个 CPU 称为 CPU 核。

1. 运算器

运算器是计算机中的执行部件,它接受控制器的命令而动作,从而完成对信息的加工和处理。运算器主要由算术逻辑单元(ALU)、累加寄存器(AC)、数据缓冲寄存器(DR)和状态条件寄存器(PSW)及通用寄存器组(REG)等组成,目前许多 CPU 还内置了协处理器,主要负责浮点运算和向量运算。

(1) 算术逻辑运算单元(Arithmetic and Logic Unit,ALU)

ALU 主要完成对二进制数据的定点算术运算(加、减、乘、除)、逻辑运算(与、或、非、异或)以及移位操作等,在某些 CPU 中还有专门用于处理移位操作的移位器。通常 ALU 有两个输入端和一个输出端。整数单元有时也称为 IEU(Integer Execution Unit),通常所说的“CPU 是 XX 位的”就是指 ALU 所能处理的数据的位数。

(2) 浮点运算单元(Floating Point Unit,FPU)

FPU 主要负责浮点运算和高精度整数运算。有些 FPU 还具有向量运算的功能,另外一些则具有专门的向量处理单元。

(3) 通用寄存器组

通用寄存器组是一组最快的存储器,用来保存参加运算的操作数和中间结果。在通用寄存器的设计上,RISC 与 CISC 有着很大的不同。CISC 的寄存器通常很少,主要是受了当时硬件成本所限,比如 x86 指令集只有 8 个通用寄存器。所以,CISC 的 CPU 执行指令时大多数时间是在访问存储器中的数据,而不是寄存器,这就拖慢了整个系统的速度。而 RISC 系统往往具有非常多的通用寄存器,并采用了重叠寄存器窗口和寄存器堆等技术使寄存器

资源得到充分的利用。

对于 x86 指令集只支持 8 个通用寄存器的缺点，Intel 和 AMD 的最新 CPU 都采用了一种叫做“寄存器重命名”的技术，这种技术使 x86 CPU 的寄存器可以突破 8 个的限制，达到 32 个甚至更多。不过，相对于 RISC 来说，采用这种技术的寄存器操作要多出一个时钟周期，用来对寄存器进行重命名。

(4) 专用寄存器

专用寄存器通常是一些状态寄存器，不能通过程序改变，由 CPU 自己控制，表明某种状态。

2. 控制器

控制器主要由程序计数器(PC)、指令寄存器(IR)、指令译码器、地址译码器、时序发生器和操作控制器等组成。控制器是计算机中发布命令的“决策机构”，即完成协调和指挥整个计算机系统的操作。它接受来自主存储器的指令，根据各条机器指令的不同功能和要求，正确地、严格地按照一定的时间次序为各个部件提供操作控制信号，并控制各个寄存器之间、CPU 同主存及 I/O 设备之间的数据流向。

信息要在不同部件之间流动，就必须有传送的通路。多个寄存器之间传送信息的路径称为**数据通路**。信息从什么地方开始，中间经过哪个寄存器或多路开关，最后传送到哪个寄存器，都要加以控制。在各寄存器之间建立数据通路的任务，是由称为操作控制器的部件来完成的。操作控制器的功能就是根据指令操作码和时序信号，产生各种微操作控制信号，以便正确地建立数据通路，从而完成取指令和执行指令的控制。

根据操作控制器的组成及产生微操作控制信号方法的不同，控制器可分为

(1) 硬布线控制器，采用组合逻辑技术或门阵列实现。

(2) 微程序控制器，采用存储逻辑实现。

无论是硬布线控制器还是微程序控制器，它们所给出的控制信号都是在一个称为时序发生器的部件所产生的连续时序信号的同步下产生的，由这些控制信号控制计算机的各个部件有条不紊地工作。

随着超大规模集成电路(VLSI)技术的发展，许多早期放在 CPU 外部的功能部件被集成到 CPU 内部，使得 CPU 的内部结构越来越复杂。现在通用的 CPU 中除集成了运算器、控制器部件，还集成了高速缓存(cache)。近年来，在一个芯片上集成了多个处理器核心称为多核处理器，有关多核处理器的内容请参考 8.5 节。

CPU 内置 cache 的容量和结构对 CPU 的性能影响很大，一般容量越大越好。在许多高性能处理器内部，一级缓存通常设置为两个，一个指令 cache，一个数据 cache，以减少取指令和读操作数的访问冲突，这种结构也叫**哈佛结构**。

Intel 80386 CPU 不包含片内 cache，但首次在 CPU 外部(主板上)使用了 cache 技术。80486 CPU 速度的提高使外部总线成为处理器访问外部 cache 的瓶颈，于是在 486 片内使用了一个小容量的 cache(8KB)；Pentium 系列 CPU 片内则集成了一级、二级甚至三级 cache，其容量也越来越大。有关 Pentium 系列 CPU 片内 cache 的详细内容请参考 4.4.6 小节。

为方便读者建立计算机整机概念，对 CPU 内部结构进行了一定的简化，给出如图 7-1 所示的 CPU 模型。

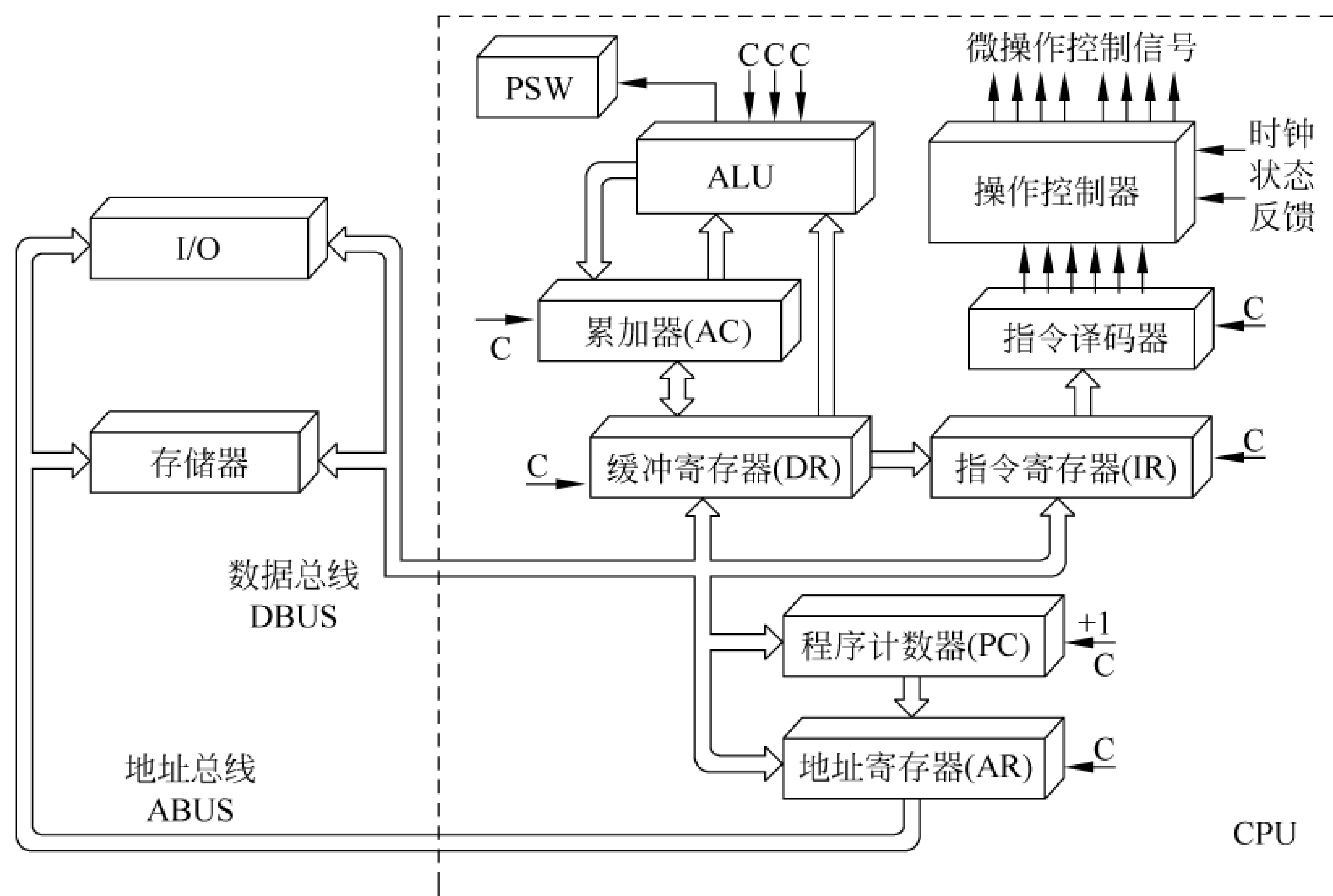


图 7-1 CPU 模型

7.1.3 CPU 的寄存器组织

计算机中传送的信息包括控制信息和数据信息,而寄存器就是用来暂存这些信息的部件。在 CPU 内部有多个寄存器,有的用于数据处理,有的用于控制,还有的用作 CPU 与主存、I/O 接口间传送信息。下面将详细介绍这些寄存器的功能与结构。

1. 累加寄存器

累加寄存器(Accumulator,AC)通常简称为累加器,它是一个通用寄存器,其功能是:当运算器的算术逻辑单元(ALU)执行算术或逻辑运算时,为 ALU 提供一个工作区。累加寄存器可提供操作数,也可存放运算结果。显然,运算器中至少要有一个累加寄存器。

目前 CPU 中的累加寄存器,多达 16 个,32 个,甚至更多。当使用多个累加器时,就变成通用寄存器结构,其中任何一个都可存放源操作数,也可存放运算结果。在这种情况下,需要在指令格式中对寄存器进行编址。通用寄存器是一组用户编程时可访问的、具有多种功能的寄存器,在指令系统的说明资料中会说明这组寄存器的存在与基本用途。

有的计算机将这组寄存器设计成基本通用,如 PDP-11 中的通用寄存器组命名为 R_0、R_1、R_2、…,它们可被指定担任各种工作,大部分寄存器没有特定的分工。有的计算机则为这组寄存器分别规定某一基本任务,并按各自的基本任务命名,如 Intel 80x86 CPU 设置有:累加器(AC)、基址寄存器(BR)、计数寄存器(CR)、数据寄存器(DR)等。

2. 用于控制的寄存器

(1) 指令寄存器(Instruction Register,IR)

指令寄存器用来保存当前正在执行的一条指令。当执行一条指令时,先把它从内存中取出并放到缓冲寄存器中,然后再传送至指令寄存器。指令划分为操作码和地址码字段,由

二进制代码表示。为了执行任何给定的指令，必须对操作码进行测试，以便识别所要求的操作。指令译码器就是做这项工作的。指令寄存器中操作码字段的输出就是指令译码器的输入，操作码一经译码后，即可向操作控制器发出具体操作的特定信号。

为了提高读取指令的速度，常在主存的数据寄存器与指令寄存器间建立直接传送通路。为了提高指令间的衔接速度，大多数计算机都将指令寄存器扩充为指令队列，允许预取若干条指令。

(2) 程序计数器(Programme Counter，PC)

为了保证程序能够连续地执行下去，CPU 必须通过某些手段来确定下一条指令的地址。而程序计数器(PC)正是起到这种作用的，所以通常又称为**指令计数器**。在程序开始执行前，必须将它的起始地址，即程序的第一条指令所在的内存单元地址送入 PC，因此 PC 的内容即是从内存提取的第一条指令的地址。当程序流程为顺序执行时，每读取一条指令后，程序计数器的内容就增量计数，以指向后继指令的地址。若主存按字节编址，则增量值取决于指令字节数，例如每读取一条单字节指令，PC 值相应加 1；如果读取一条二字节指令，则 PC 加 2。

但是，当遇到转移指令(如 JMP 指令)时，那么后继指令的地址(即 PC 的内容)必须从指令的地址字段取得。在这种情况下，下一条从内存取出的指令将由转移指令来规定，而不是像通常一样按顺序来取得。因此程序计数器应当是具有计数和寄存信息两种功能的寄存器。

(3) 状态条件寄存器(Programme Status Word，PSW)

CPU 的主要工作是执行程序，其过程一方面取决于编程时的程序流向，另一方面也取决于程序实际执行的状态。因此，许多计算机设置有一个状态条件寄存器，其内容表示 CPU 当前的基本状态，也就是现行程序的状态，常称为程序状态字(PSW)。其中一部分内容记载程序运行的状态，另一部分内容可由编程设定，如设置工作方式等。

PSW 中的若干位可用来记录程序执行状态，称为标志位或特征位。一条指令执行后，将根据运行结果自动修改标志位的有关内容，作为决定程序流向的因素之一，常见的标志位有：

① 进位标志 C，表示运算后是否有进位产生。

② 溢出标志 OV，表示运算结果是否有溢出。

③ 全零标志 Z，表示运算结果是否为零。

④ 符号标志 N，表示运算结果是否为负数。

⑤ 奇偶标志 P，表示运算结果中 1 的个数是奇数还是偶数。

有的计算机还设置有中断允许(IF)、半进位标志(AF)、单步标志(TF)、方向标志(DF)(地址由低到高，还是由高到低)等。

3. 用于主存接口的寄存器

当 CPU 访问主存或 I/O 接口时，均应首先送出地址码，然后再读、写数据。为此，常设置以下两个寄存器。

(1) 地址寄存器

地址寄存器(Address Register，AR)用来保存当前 CPU 所访问的内存单元的地址。由

于在内存和 CPU 之间存在着操作速度上的差别，所以必须使用地址寄存器来保持地址信息，直到内存的读/写操作完成为止。

当 CPU 和内存进行信息交换，即 CPU 从内存读/写数据时，或者 CPU 从内存中取指令时，都要使用地址寄存器和数据缓冲寄存器。同样，如果把外围设备的设备地址作为像内存单元的地址那样来看待，那么，当 CPU 和外围设备交换信息时，同样要使用地址寄存器和数据缓冲寄存器。

(2) 数据缓冲寄存器

数据缓冲寄存器(Data Register，DR)用来暂时存放由内存读出的一条指令或一个数据；反之，当向内存存入一个数据时，也暂时将它们存放在数据缓冲寄存器中。

缓冲寄存器的作用是：

① 作为 CPU 和内存、外部设备之间信息传送的中转站；

② 补偿 CPU 和内存、外围设备之间在操作速度上的差别；

③ 在单累加器结构的运算器中，数据缓冲寄存器还可兼作操作数寄存器。

设置 AR 与 DR，使 CPU 与主存间的传送通路变得比较单一，容易控制。对用户来说，这两个寄存器往往是“透明”的，不能直接编程访问。

地址寄存器(AR)、数据缓冲寄存器(DR)、指令寄存器(IR)的结构一样，通常使用单纯的寄存器结构。信息的存入一般采用电位-脉冲方式，即电位输入端对应数据信息位，脉冲输入端对应控制信号，在控制信号的作用下，实时地将信息打入寄存器。

CPU 中除了上述组成部分外，还有中断系统、总线接口等其他功能部件，这些内容将在其他章节进行详细讨论。

7.2　指令周期

CPU 对指令的执行必须有条不紊，为此，指令被分成一个个操作步，称为微操作。每一个微操作被分配到一个时间单元中执行，以保证它们在执行上的顺序关系。在 7.2 节，先介绍一些时间概念，然后以一个模型机为例，介绍一些典型指令如何进一步划分成一个个微操作以及这些微操作的执行过程。

7.2.1　几个时间概念

计算机之所以能自动工作，是因为预先编写好了解决问题的程序并把它存放在内存中。开始工作后，CPU 能按要求从内存中取出一条指令并执行这条指令；紧接着又取下一条指令，执行下一条指令……如此周而复始，构成了一个封闭的循环。除非遇到停机指令，否则这个循环将一直继续下去，其过程如图 7-2 所示。

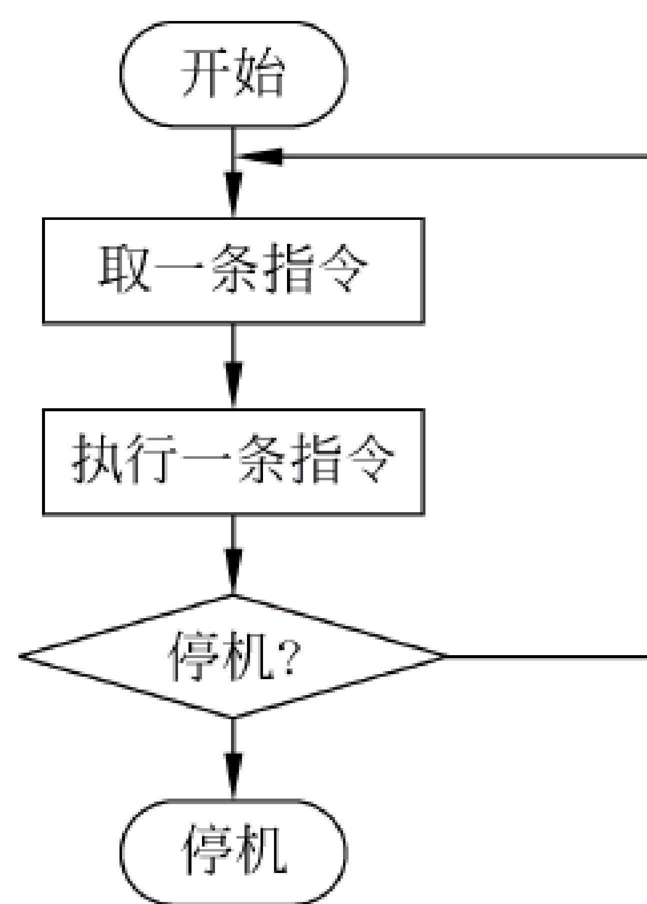


图 7-2　指令处理过程

指令的处理过程需要一定的时间，取指令需要时间，指令的执行也需要时间，通常将 CPU 从内存中取出一条指令并执行这条指令的时间总和称为**指令周期**。

由于各机器指令的功能不同，指令中操作数所采用的寻址方

式也不尽相同，故不同指令执行的时间长短是不相同的，即不同机器指令的指令周期各不相同。如同样是加法指令，操作数采用寄存器寻址和直接寻址所花的时间就不相同。

指令周期通常用**机器周期**来描述，机器周期又称为 **CPU 周期**。指令执行过程中每一步操作（如取指令）所需的时间对应着一个 CPU 周期，由于 CPU 访存的时间相对 CPU 内部操作的时间较长，常将从内存读出一个指令的最短时间作为一个 CPU 周期，它代表了大多数指令操作步骤的时间。

而一个 CPU 周期又包含若干个**时钟周期**，如图 7-3 所示。时钟周期也叫做节拍脉冲或 *T* 周期，它对应着计算机中最基本的定时信号，由它可推出计算机的主时钟频率。

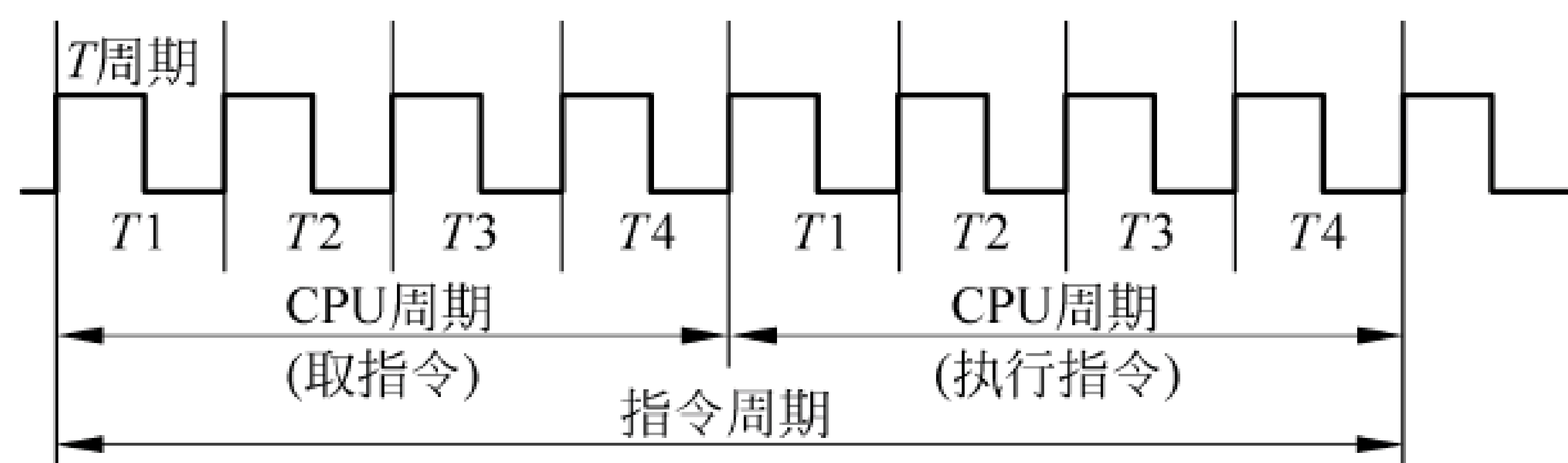


图 7-3　指令周期

应当指出，在不同的计算机中对 CPU 周期的规定是不尽相同的，有的指令周期包含的 CPU 周期数多，有的则较少。有的机器中一个 CPU 周期中所包含的时钟周期数是固定的，这叫定长机器周期，如图 7-3 所示；而有的机器则采用不定长的机器周期，如图 7-4 所示。

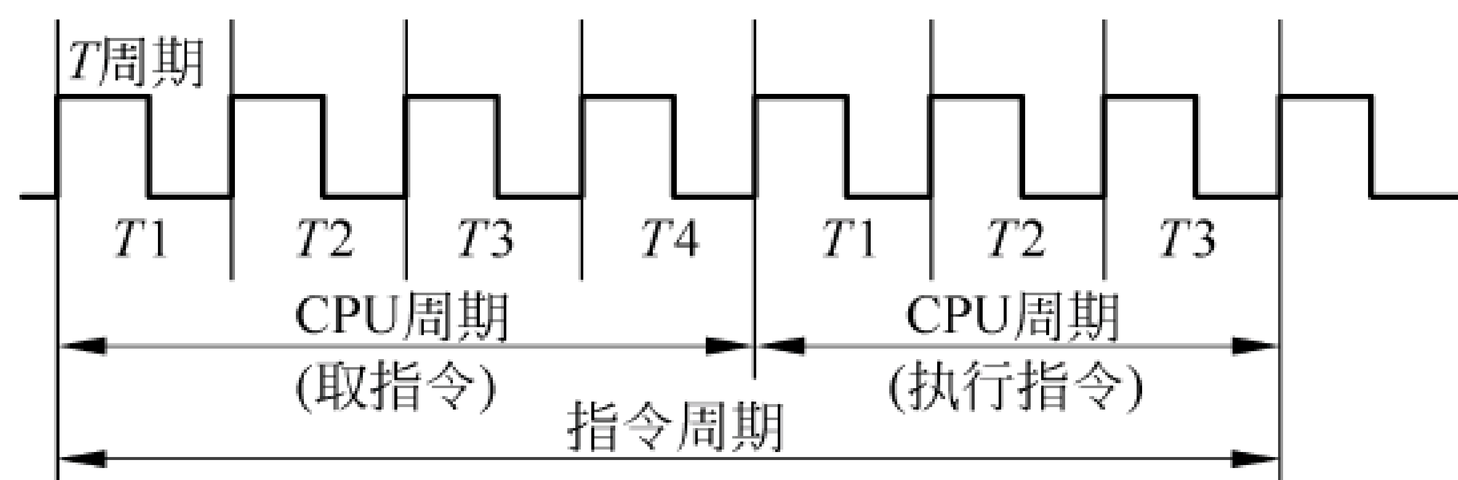


图 7-4　不定长的机器周期

（1）取指周期。

取指周期是每条指令都必须有的。取指周期要完成两项任务，一是要以 PC 的内容为地址从主存中取出指令代码送 IR，二是要修改 PC 形成后继指令地址。但是，当执行转移指令时，PC 中的地址在转移指令的执行阶段修改。

（2）取操作数周期。

这个机器周期的任务是形成操作数地址并获取操作数，操作数地址的形成与寻址方式有关，提供操作数的部件可以是寄存器，也可以是存储器，因此，取操作数的操作因指令而不同。有些指令的执行过程是不需要取操作数的，因此，这个机器周期不是每条指令都必须有的。

（3）执行周期。

执行周期的主要任务是完成指令操作码规定的操作，并记录指令执行后的状态信息。执行周期的操作对不同的指令也是不相同的。

综上所述，一条指令的指令周期至少包括两个机器周期，不同的指令需要的机器周期数是不同的。

【例 7.1】 某计算机执行一条指令平均需要 3 个 CPU 周期，平均需访问内存 2 次。每个 CPU 周期包含 4 个时钟周期。若计算机主频为 100MHz，问：

(1) 若主存为"0 等待"(即不需插入等待周期)，问指令执行的平均速度为多少?

(2) 若每次访问内存需插入一个等待周期，问执行一条指令的平均时间为多少？指令执行的平均速度又是多少?

解：计算机主频为 100MHz，所以时钟周期为

$$T_{CLK} = \frac{1}{100 \times 10^6 \text{Hz}} = 1 \times 10^{-8} \text{s} = 10\text{ns}$$

(1) 因为每个 CPU 周期包含 4 个时钟周期，所以 CPU 周期为

$$T_{CPU} = 4 \times T_{CLK} = 40\text{ns}$$

所以，平均速度为

$$\frac{1}{3T_{CPU}} = \frac{1}{3 \times 40\text{ns}} = \frac{1}{3 \times 40 \times 10^{-9}\text{s}} \approx 8.33\text{MIPS}$$

(2) 执行一条指令平均需要 3 个 CPU 周期，需访问内存 2 次，且每次访问内存需插入一个等待周期，所以执行一条指令的平均时间为

$$3T_{CPU} + 2T_{CLK} = 3 \times 40\text{ns} + 2 \times 10\text{ns} = 140\text{ns}$$

平均速度为

$$\frac{1}{3T_{CPU} + 2T_{CLK}} = \frac{1}{140\text{ns}} \approx 7.14\text{MIPS}$$

7.2.2 典型指令的指令周期

表 7-1 给出了 4 条典型指令组成的一个简单程序，其中 LDA 是数据传送指令，ADD 是加法指令，STR 是存操作数指令，JMP 是无条件转移指令，实现程序顺序控制。7.2.2 节通过每一条指令取指阶段与执行阶段的分解动作，来具体认识每一条指令的指令周期。

表 7-1　典型指令组成的一个简单程序

十六进制数地址	十六进制数代码	助记符	指令功能说明
0010	3E00	LDA 00H	AC←00H
0011	18E0	ADD [E0H]	AC←(AC)+(00E0H)
0012	10F0	STR [F0H]	00F0H←(AC)
0013	6011	JMP 11H	PC←0011H
…	…	…	…
00E0	0005	数据	
00E1	0020		
…	…		
00F0	存放和数		

1. LDA 指令的指令周期

假定表 7-1 的简单程序已装入内存中，程序计数器(PC)指向程序的首地址，PC=

0010H，即存放 LDA 00H 指令的内存单元，所以 CPU 首先执行的第一条指令是 LDA。LDA 00H 采用立即寻址，执行阶段不要访问内存，其功能是把立即数传送到累加器，其指令周期如图 7-5 所示。它需要两个 CPU 周期，其中取指令阶段需要一个 CPU 周期，执行指令阶段需要一个 CPU 周期。

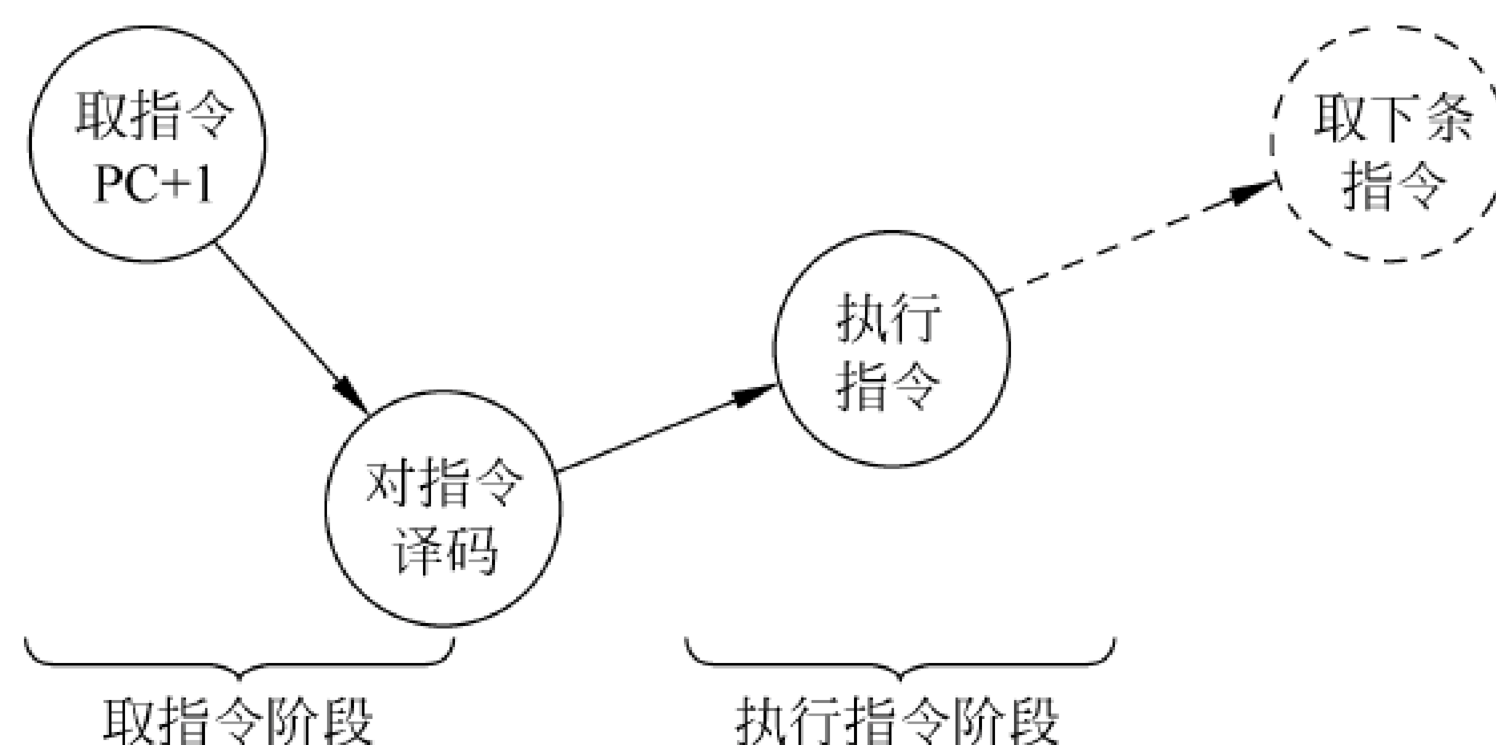


图 7-5　LDA 的指令周期

第一个 CPU 周期是取指周期，CPU 完成三件事：①从内存取出指令。②对程序计数器 PC 加 1，为取下一条指令做好准备。③对指令操作码进行译码或测试，以便确定进行什么操作。

在第二个 CPU 周期，即执行指令阶段，CPU 根据对指令操作码的译码得到的操作控制信号，进行指令所要求的操作。对非访内指令来说，执行阶段通常涉及对累加器进行操作，如累加器内容清零、求反等操作。显然，其他一些零地址格式的指令，执行阶段也仅需要一个 CPU 周期。

1）取指周期

LDA 的取指周期操作如图 7-6 所示，在此阶段内，CPU 的操作如下：

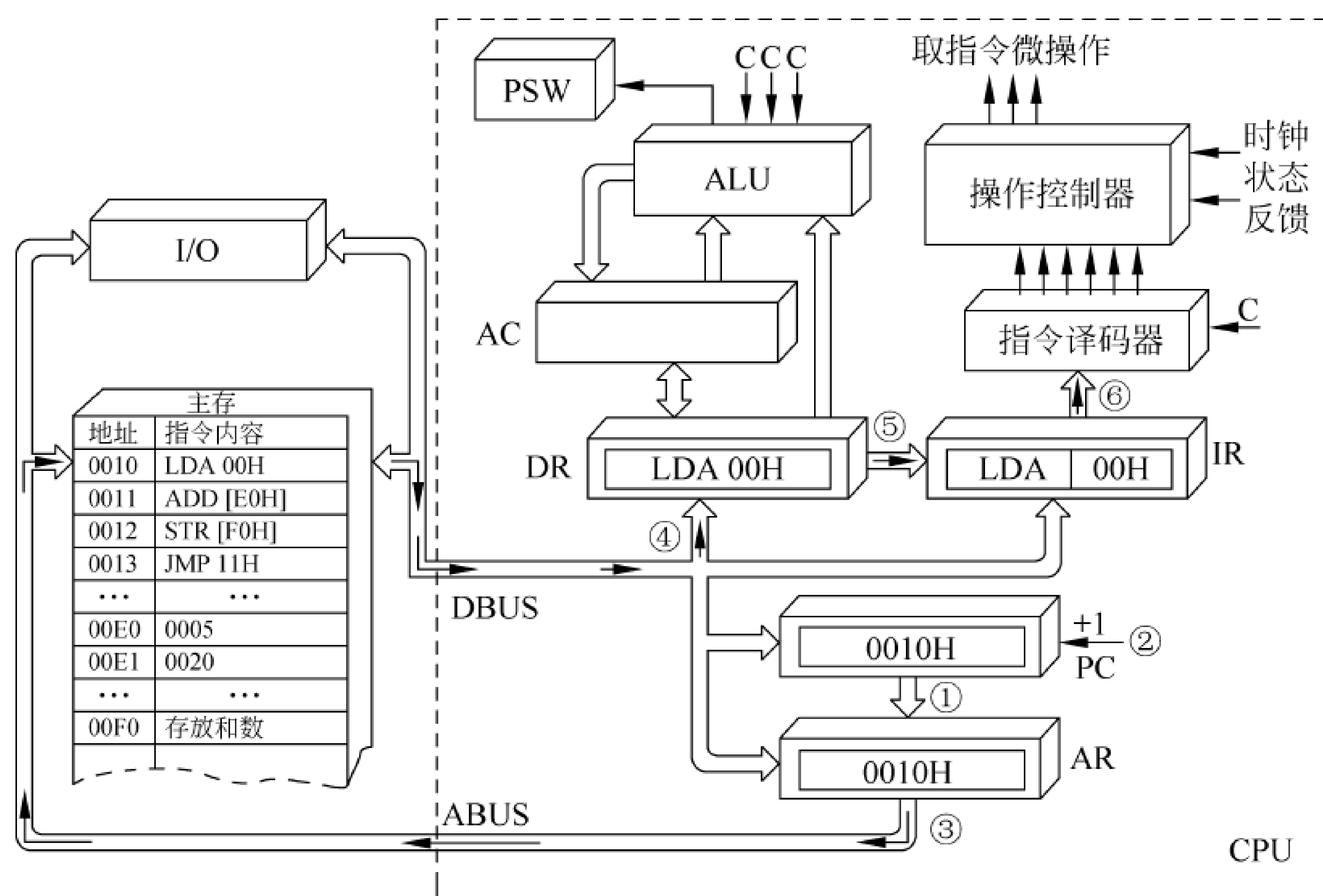


图 7-6　LDA 00H 的取指周期操作

① 程序计数器(PC)的内容 0010H 被装入地址寄存器(AR)。

② 程序计数器内容加 1,变成 0011H,为取下一条指令做好准备。

③ 地址寄存器的内容被放到地址总线上。

④ 所选存储器单元 0010H 的内容经过数据总线,传送到数据缓冲寄存器(DR)。

⑤ 缓冲寄存器的内容传送到指令寄存器(IR)。

⑥ 指令寄存器中的操作码被译码或测试。

⑦ CPU 识别出是指令 LDA,至此,取指令阶段即告结束。

需要说明的是,取指周期的过程和数据通路对每条指令来说都是一样的,所不同的是 PC 的值和取得的指令代码不一样。后续指令 ADD、STR、JMP 的取指周期与 LDA 指令相同,因此不必重复讨论。

2）执行指令阶段

LDA 指令的执行阶段如图 7-7 所示,在此阶段内,CPU 的操作如下:

① 经译码后知道 LDA 00H 指令采用立即寻址方式,操作数跟在操作码之后,已经存放在指令寄存器(IR)中,操作控制器送出控制信号,在 IR 与数据缓冲寄存器(DR)之间建立数据通路,把立即数打入 DR。

② 操作控制器再送出控制信号在 DR 与 ALU 之间建立数据通路,把立即数送到 ALU 的输入端。

③ ALU 响应传送操作控制信号,将输入 ALU 的数据送到累加寄存器(AC),从而执行完了 LDA 00H 指令。

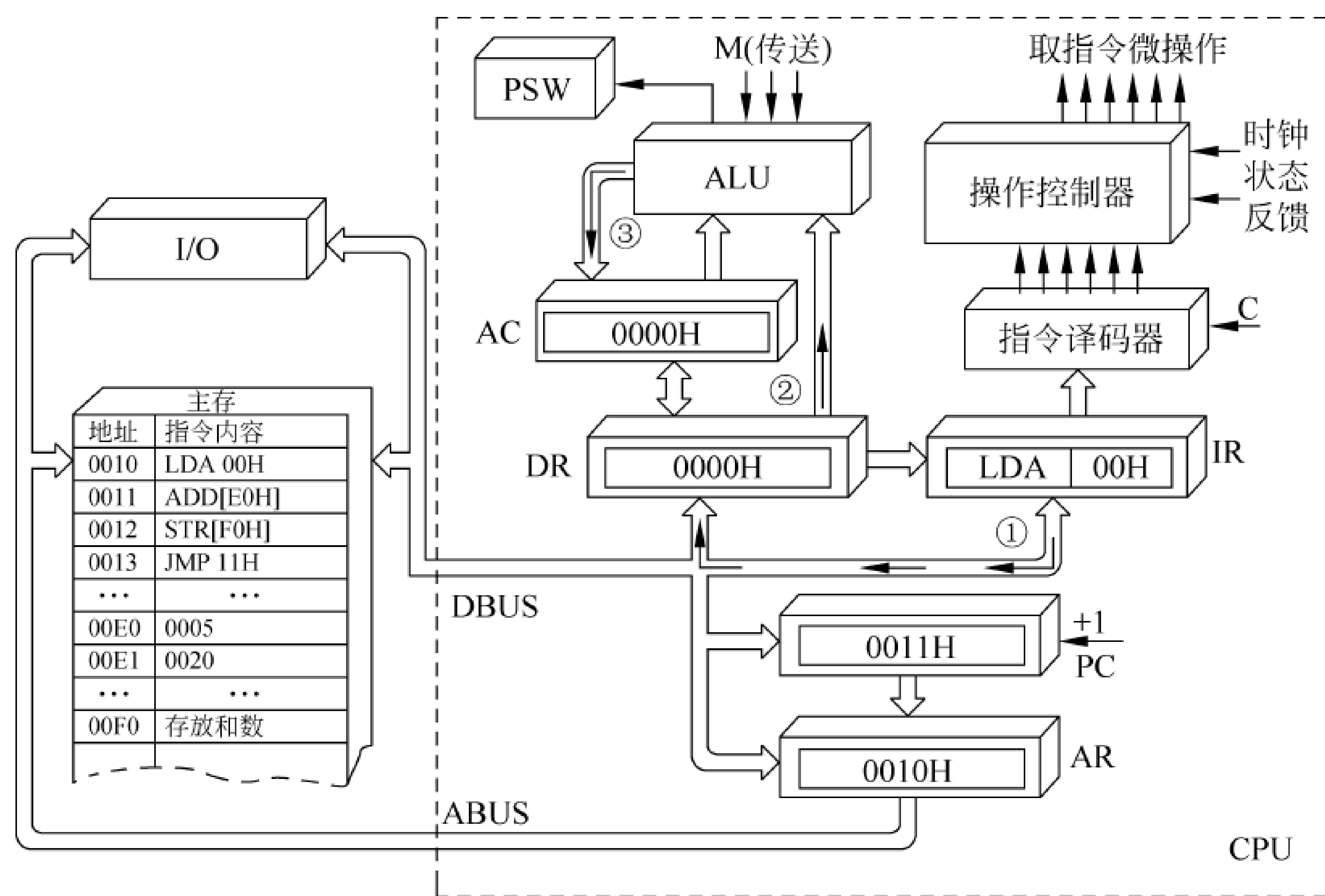


图 7-7　LDA 指令执行阶段

2. ADD 指令的指令周期

取出第一条指令 LDA 时,PC 的内容已经加 1 变成了 0011H,这个地址指向存放 ADD [E0H]指令的内存单元,所以接下来执行的指令是 ADD [E0H]。

ADD 指令是一条访问内存取数并执行加法的指令,它的指令周期由三个 CPU 周期组成,如图 7-8 所示。其中第一个 CPU 周期为取指令阶段,执行阶段由两个 CPU 周期组成,其中在第二个 CPU 周期中计算并送操作数地址,第三个 CPU 周期读操作数执行加法操作。

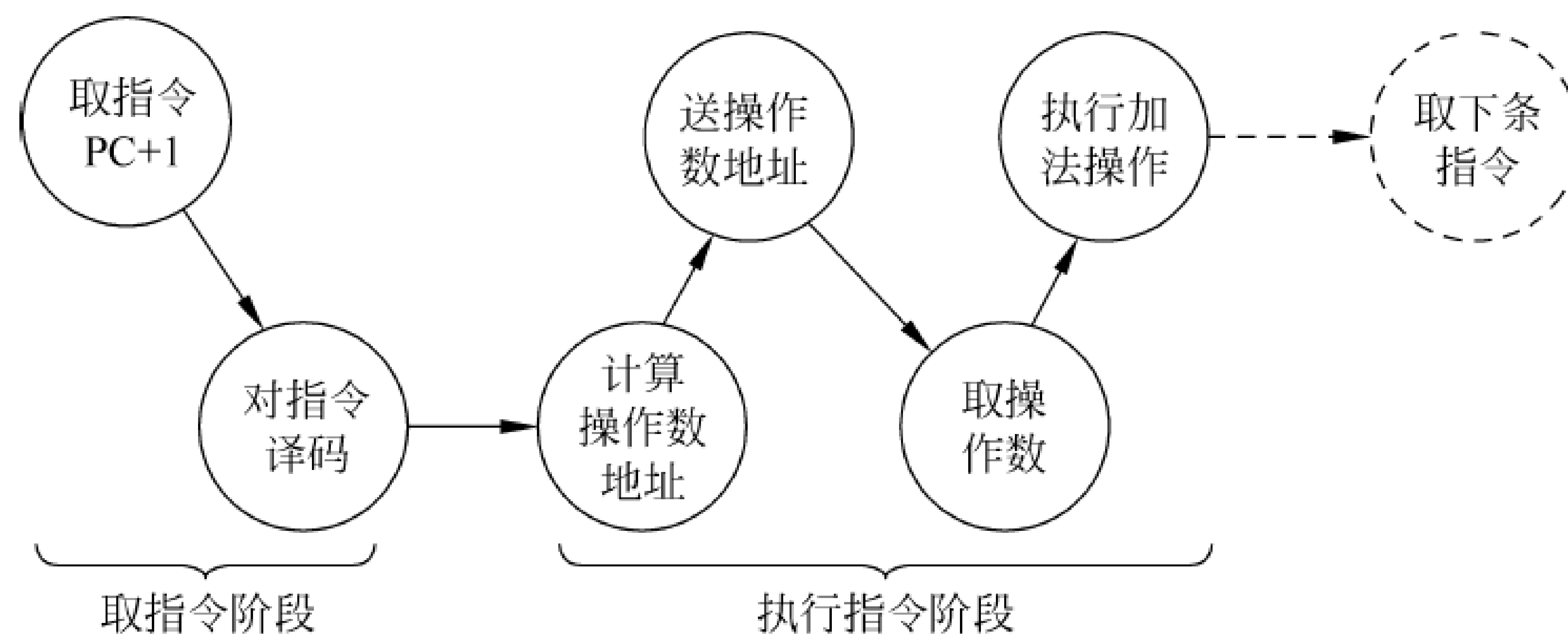


图 7-8 ADD 的指令周期

1）取指令阶段

ADD 指令的第一个 CPU 周期完成取指令操作并对指令译码,ADD 指令取出后,PC＝0012H。

2）计算并传送操作数地址

第二个 CPU 周期主要根据寻址方式计算操作数地址,传送操作数地址。此阶段的操作如图 7-9 所示,在此阶段内,CPU 的操作如下:

（1）ADD [E0H]是直接寻址,操作数地址为 00E0H。

（2）把指令寄存器中的地址码部分(E0H)装入地址寄存器(AR)。

（3）把地址寄存器中的操作数地址(00E0H)发送到地址总线上。

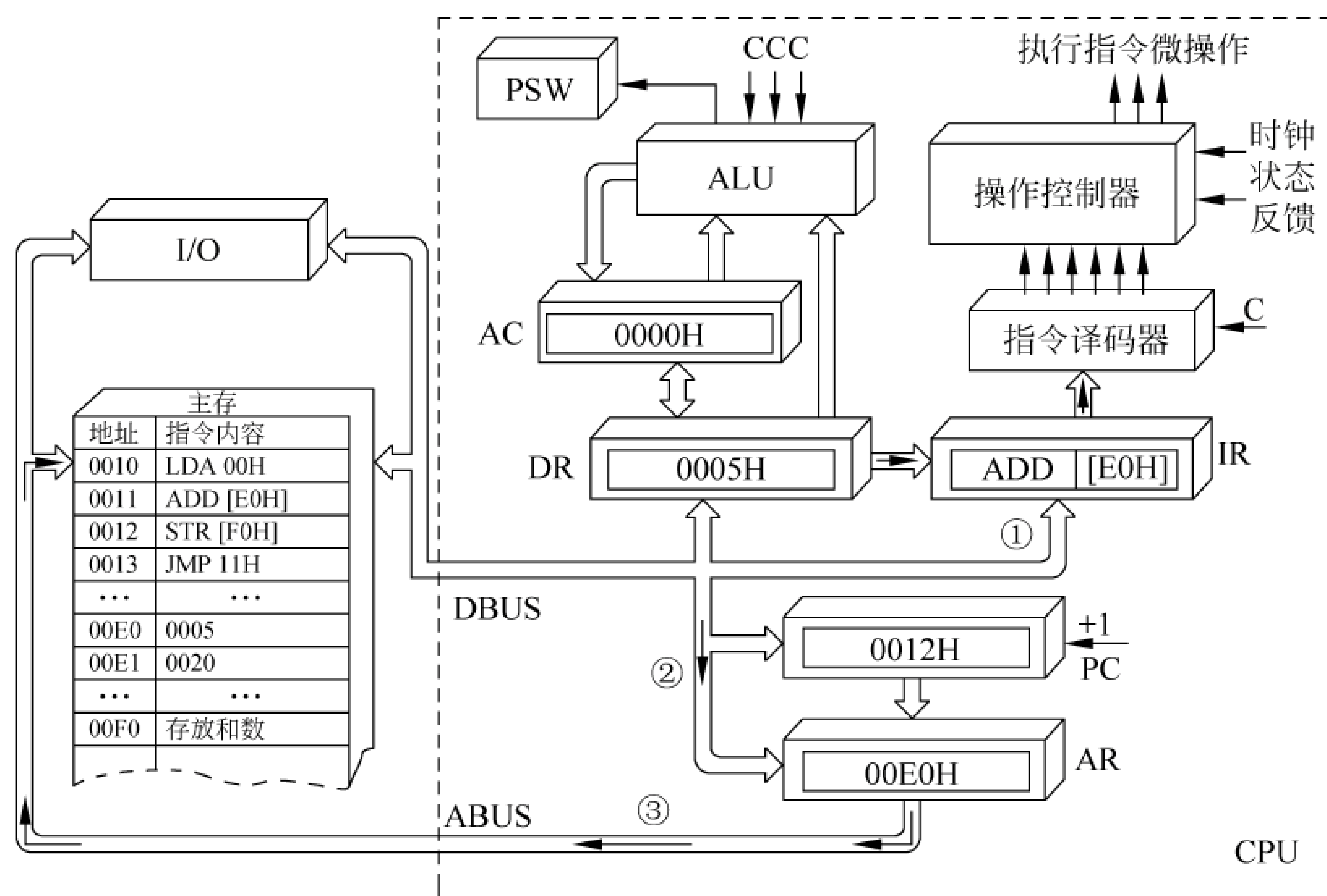


图 7-9 ADD 指令计算并传送操作数地址

3）读操作数执行加法操作

第三个 CPU 周期主要完成加法操作，如图 7-10 所示，在此阶段，CPU 完成以下动作：从存储器单元(00E0H)中读出操作数(0005H)，并经过数据总线传送到缓冲寄存器。由数据缓冲寄存器来的操作数(0005H)可送往 ALU 的一个输入端，已等候在累加器内的另一个操作数(因为 LDA 00H 指令执行结束后累加器内容为零)送往 ALU 的另一输入端，于是 ALU 将两数相加，产生运算结果为 0+0005H=0005H。这个结果放回累加器，替换了累加器中原先的数 0000H。

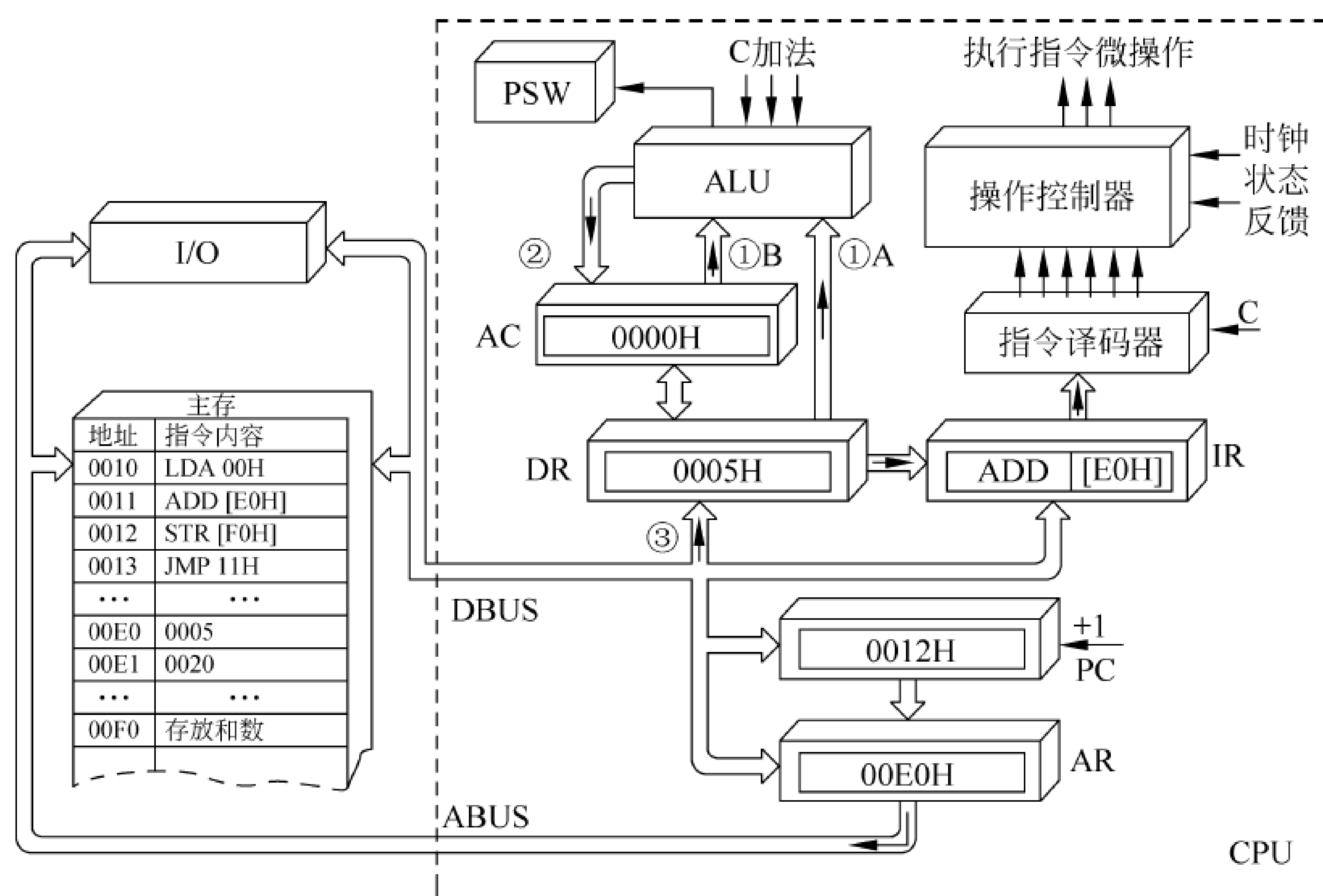


图 7-10 ADD 指令的加法操作阶段

3. STR 指令的指令周期

取出 ADD [E0H]指令时，PC 的内容已经加 1 变成了 0012H，这个地址指向存放 STR [F0H]指令的内存单元，所以接下来执行的指令是 STR [F0H]。

STR [F0H]是一条直接寻址指令，把累加器(AC)的内容存入内存单元，它的指令周期由三个 CPU 周期组成，如图 7-11 所示。其中第一个 CPU 周期为取指令阶段，第二个 CPU 周期中计算并送操作数地址，第三个 CPU 周期写操作数进存储单元。

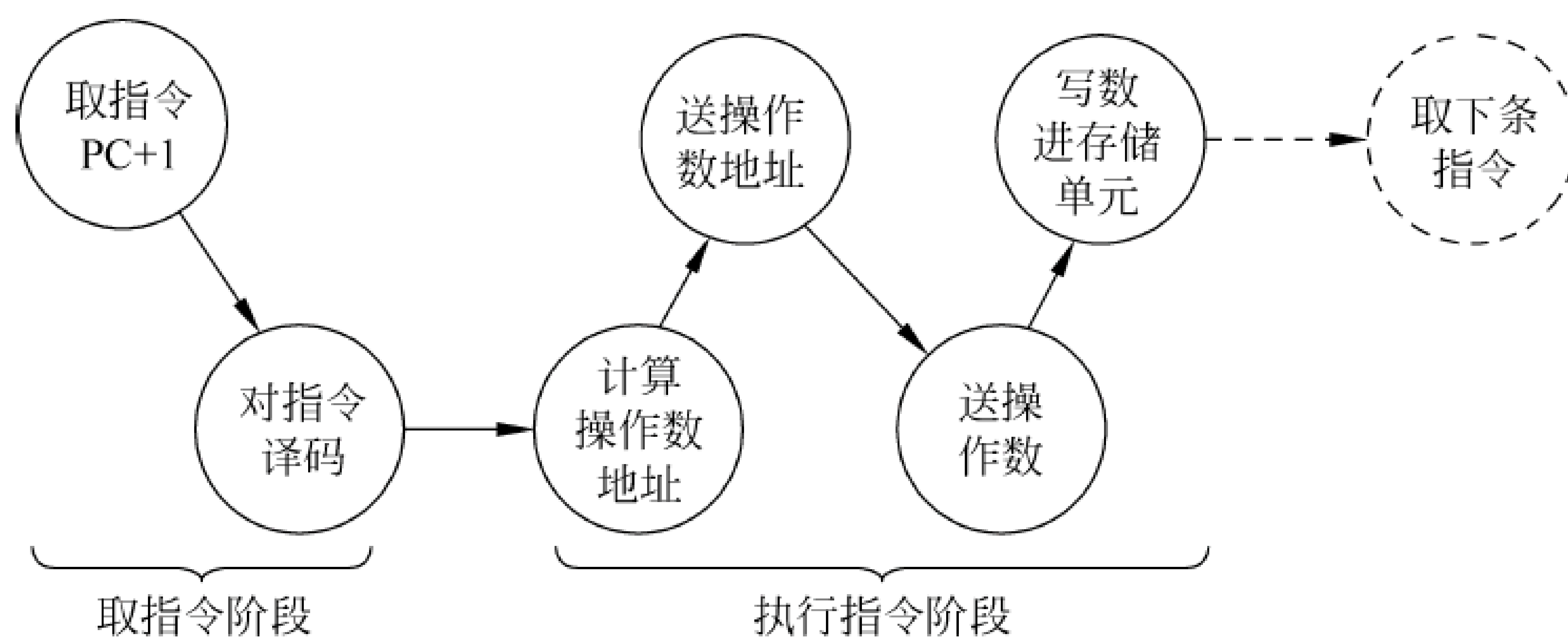


图 7-11 STR 的指令周期

1）取指令阶段

STR [F0H]指令的第一个 CPU 周期完成取指令操作并对指令译码。STR 指令取出后，PC=0013H，为取下一条指令做好了准备。

2）计算并传送操作数地址

STR [F0H]指令的第二个 CPU 周期与 ADD [E0H]指令的第二个 CPU 周期的操作相同，只是计算并传送的操作数地址码为 00F0H。

3）写操作数进存储单元

第三个 CPU 周期主要完成送操作数地址，访问内存单元存操作数。此阶段的操作如图 7-12 所示，在此阶段内，CPU 的操作如下：

（1）累加器 AC 的内容（0005H）被传送到数据缓冲寄存器（DR）。

（2）把缓冲寄存器（DR）的内容（0005H）发送到数据总线上。

（3）操作控制器发出写 WR 控制信号，将 0005H 写入存储器 00F0H 单元中。

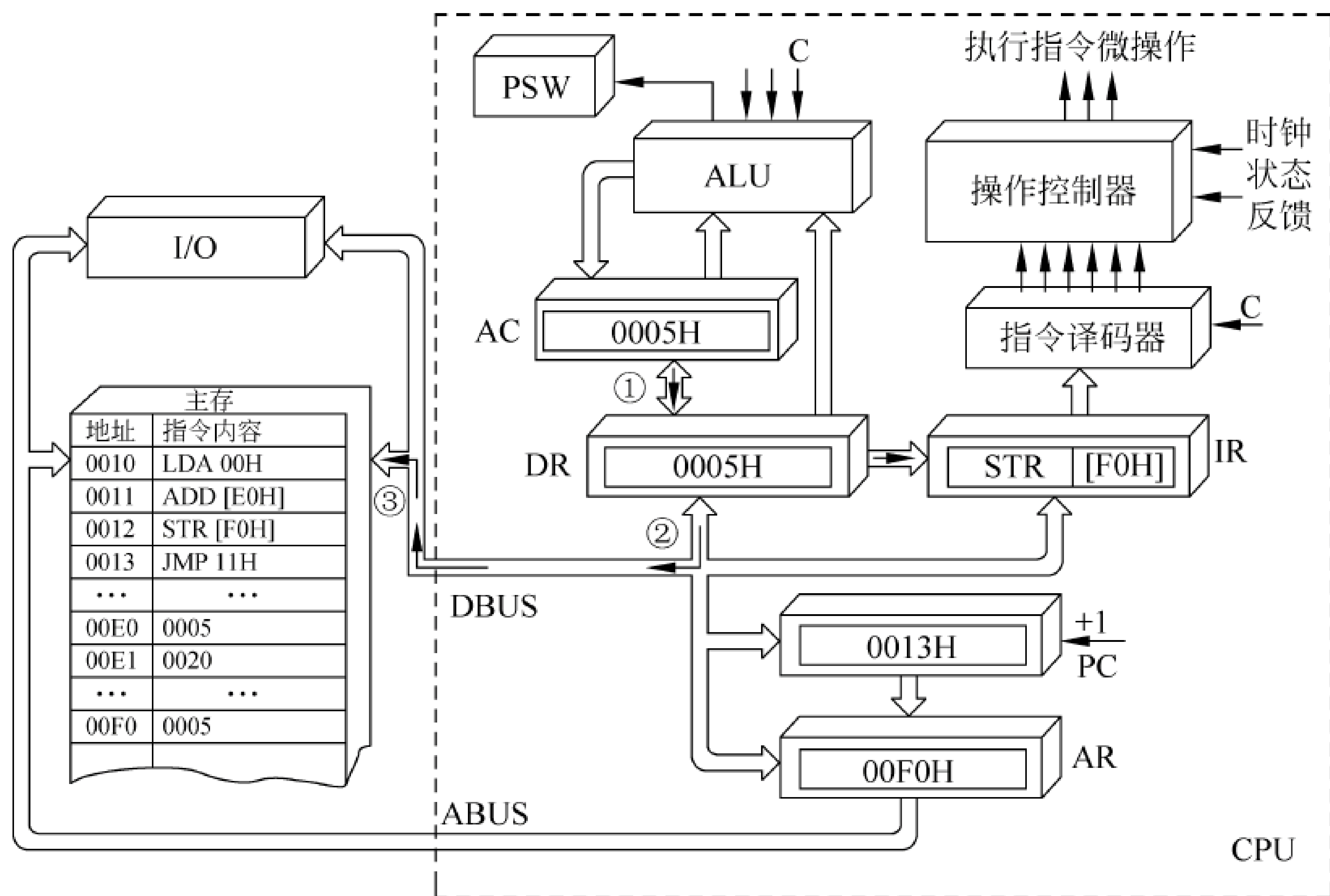


图 7-12 STR 指令写操作数进存储单元

注意：*在这个操作之后，累加器中仍然保留和数 0005H，而存储器 00F0H 单元中原先的内容被修改。*

4. JMP 指令的指令周期

STR [F0H]指令取出后，PC=0013H，因此下一条要执行的指令是存放在 0013H 单元的 JMP 11H 指令。JMP 是一条程序控制转移指令，其指令周期由两个 CPU 周期组成，如图 7-13 所示。其中第一个 CPU 周期为取指令阶段，第二个 CPU 周期为执行阶段。

1）取指令阶段

CPU 把 0013H 号单元的 JMP 11H 指令取出送到指令寄存器（IR），同时程序计数器（PC）的内容加 1，变为 0014H，指令译码后知道 IR 中的指令是无条件转移指令。

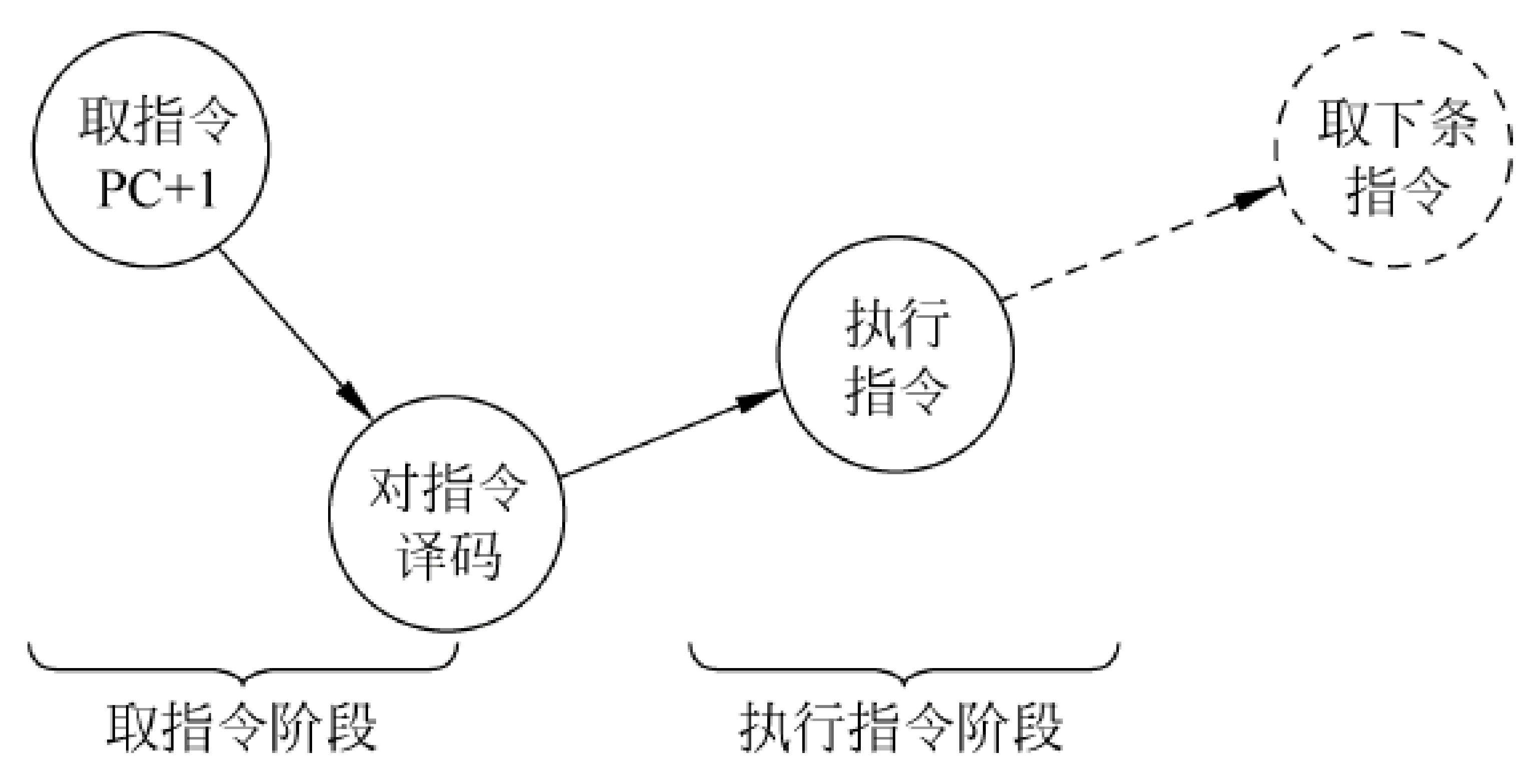


图 7-13　JMP 的指令周期

2）执行阶段

CPU 把指令寄存器（IR）中的地址码部分 0011H 送到程序计数器，从而用新内容 0011H 代替 PC 原先的内容 0014H。这样，下一条指令将不从 0014H 单元读出，而是从内存 0011H 单元开始读出并执行，从而改变了程序原先的执行顺序。JMP 指令执行阶段的操作如图 7-14 所示。

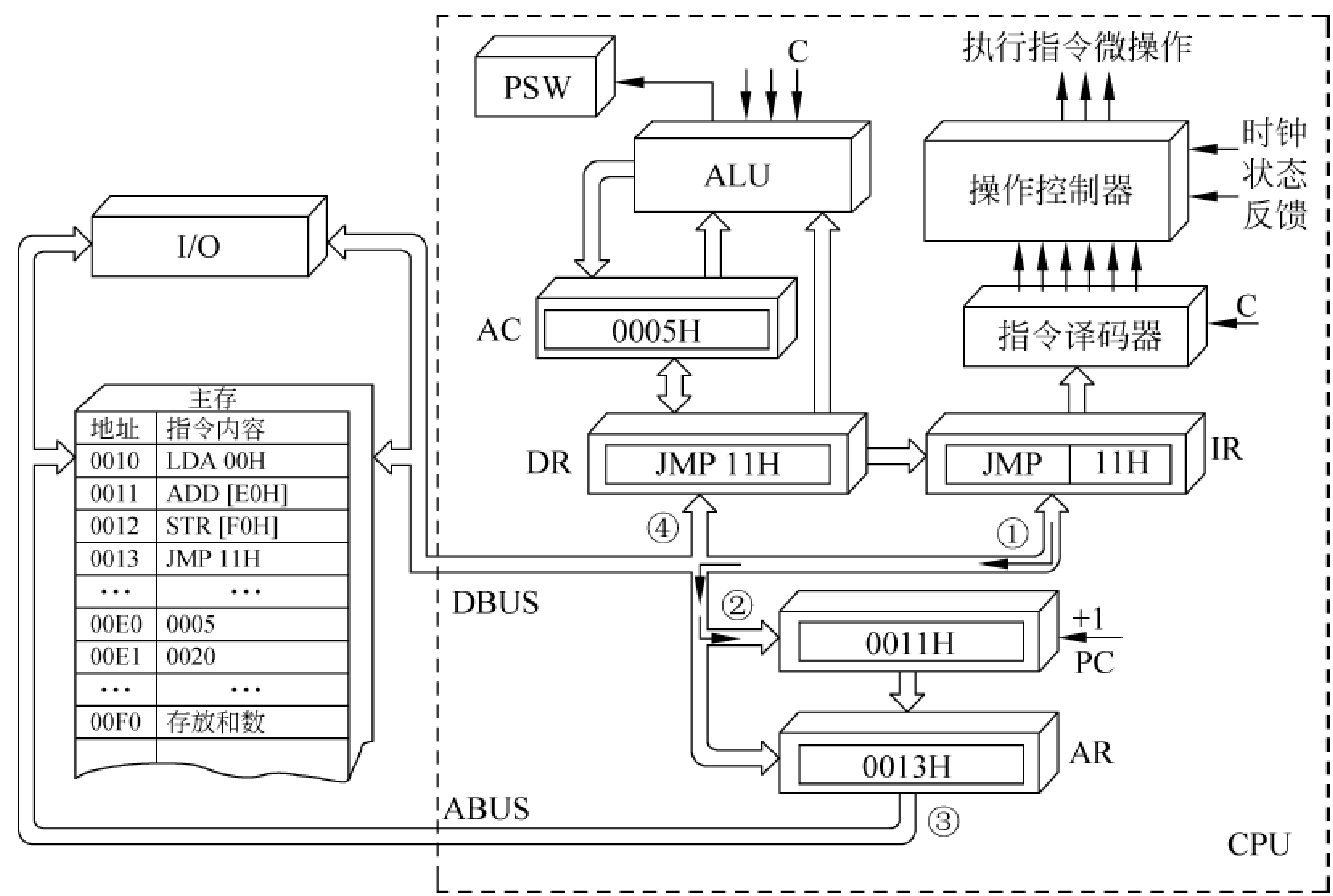


图 7-14　JMP 11H 的执行阶段

注意：执行 JMP 11H 指令时，这里所给的四条指令组成的程序进入了死循环，除非人为停机，否则这个程序将无休止地运行下去，因而内存单元 00F0H 中的和数将一直不断地发生变化。当然，此处所举的转移地址 0011H 是随意的，用来说明转移指令仅仅能够改变程序的执行顺序而已。

7.2.3　指令周期的方框图语言描述

7.2.2 节通过画 CPU 内部数据通路的方法，介绍了几条指令的指令周期，这种方法直观、形象，有利于读者理解指令的执行过程。在进行计算机设计时，如果用这种办法来表示

指令周期，那就显得过于烦琐，而且也没有必要。

在进行计算机设计时，可以采用方框图语言来表示一条指令的指令周期。一个方框代表一个CPU周期，方框中的内容表示数据通路的操作或某种控制操作。菱形框通常用来表示某种判别或测试，不过时间上它依附于紧接它的前面一个方框的CPU周期，而不单独占用一个CPU周期。

把7.2.2节的4条典型指令加以归纳，用方框图语言表示的指令周期如图7-15所示。从图7-15中可以看出，所有指令的取指令阶段是完全一样的，都是占用一个CPU周期。但是由于各条指令的功能不同，不同指令执行阶段所占用的CPU周期数是各不相同的，其中LDA、STR和JMP是一个CPU周期，ADD是两个CPU周期。

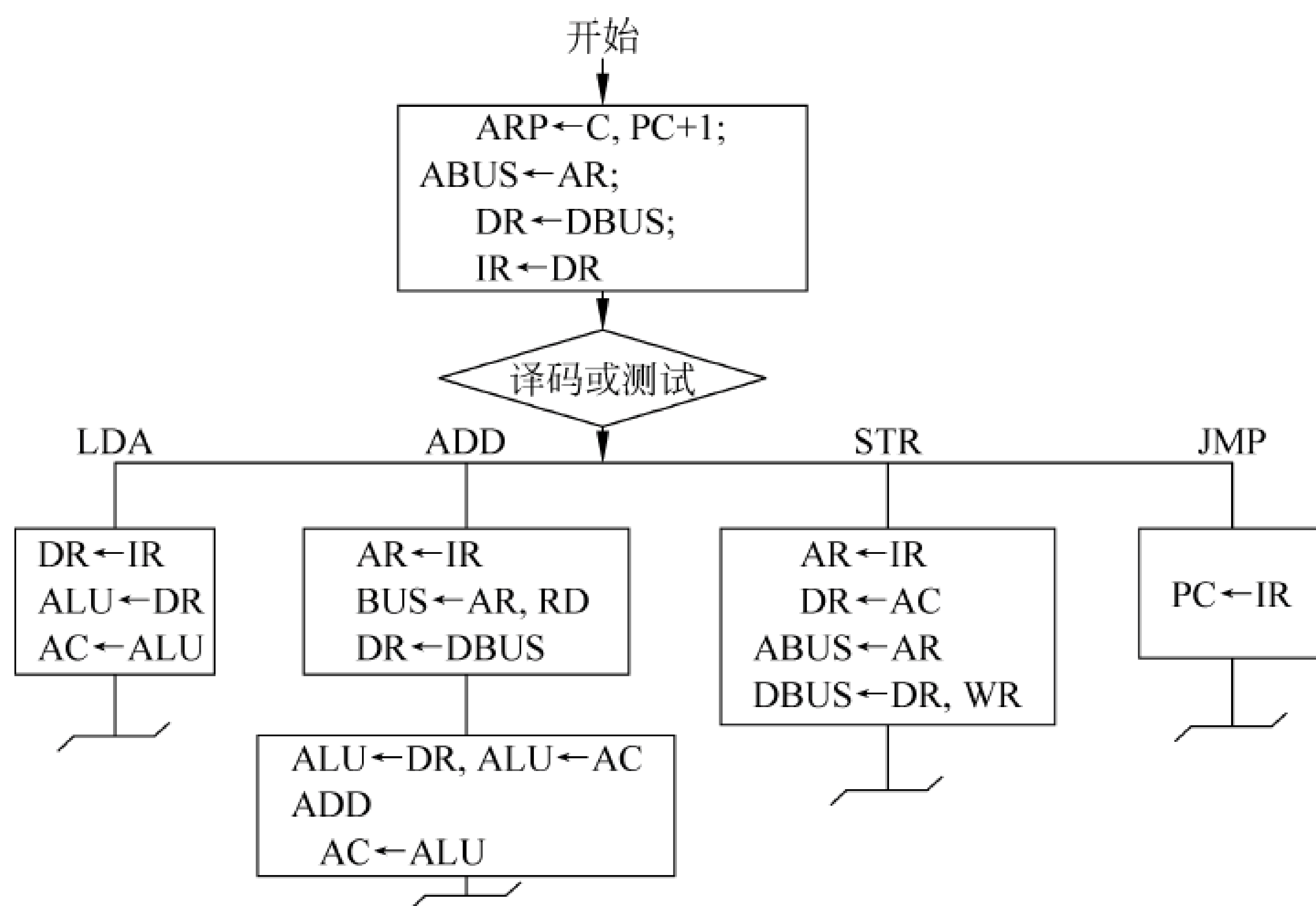


图7-15 用方框图语言表示的指令周期

图7-15中DBUS表示数据总线，ABUS表示地址总线，RD表示内存读命令，WR表示内存写命令，“ ”表示公操作符号。公操作就是一条指令执行完毕后，CPU要进行的一些操作，这些操作主要是CPU对外设请求的处理，如中断处理、通道处理等。如果没有外设请求要处理，那么CPU又转向内存取下一条指令。由于所有指令的取指令阶段完全是一样的，因此，取指令也可认为是公操作。所以，一条指令的指令周期结束后，一定会转入公操作。

7.3 CPU的时序和控制

CPU是通过指令周期-节拍-工作脉冲三级时序系统来控制指令的执行的。在本节中将主要介绍控制指令执行的时序系统和CPU的控制方式。

7.3.1 CPU的时序系统

通过7.2节的讲解已经看到，指令在CPU中执行时，是通过控制器产生的操作控制信

号去控制各个部件完成指令的功能的。由此可以看出，指令还可以再进一步细分为更小的操作步，称为微操作。**微操作**是 CPU 控制执行单元所能完成的最小操作步，如部件之间的数据传送、运算或执行部件所执行的操作等。

由于执行指令的各微操作是有先后次序的，并且许多控制信号的长短也有严格的时间限制，这就需要引入时序信号对它们进行定时控制。时序系统是控制器的心脏，其功能是为指令的执行提供各种定时信号。

7.2 节讨论了典型指令的指令周期，指令周期由若干机器周期组成。不同指令的指令周期是不完全相同的。在时序系统中一般不为指令周期设置完整的时间标志信号，因此一般不将指令周期视为时序的一级。由于 CPU 内部的操作速度快，而 CPU 访问主存所开销的时间较长，所以许多计算机系统往往以主存的存取周期为基础来规定机器周期的长度。通常，每个机器周期都有一个与之对应的周期状态触发器，机器运行在不同的机器周期时，其对应的周期状态触发器被置为“1”。显然，在机器运行的任何时刻只能处于一种周期状态，因此有一个且仅有一个周期状态触发器被置为“1”。

1. 节拍

一个机器周期内要完成若干微操作，其中有的可并行执行，有的要求顺序操作。因此，需要把一个机器周期分为若干个相等的时间段，每一时间段对应一个节拍信号，称为节拍脉冲信号。节拍的宽度取决于 CPU 完成一次基本操作的时间，如 ALU 完成一次正确的运算，寄存器间的一次传送等。

由于不同的机器周期内需要完成的微操作的个数及难易程度是不同的，因而不同机器周期内所需要的节拍数也不相同。定长机器周期以最复杂的机器周期为准来定出节拍数，每一节拍时间的长短也以最繁的微操作为准，这种方法使得所有的机器周期长度相等，并且每一机器周期内含有相同数目的节拍。不定长机器周期按照实际的需要安排节拍数，需要多少个节拍就提供多少个节拍，这种方式的时间利用率高。还有一种方法是在照顾多数机器周期要求的前提下，选取适当的节拍数作为基本节拍，若某个机器周期内按规定的基本节拍数无法完成该周期的全部微操作，则可延长节拍。

某些微型机的时序信号中不设置节拍，直接使用时钟周期信号。一个机器周期中含有若干个时钟周期，时钟周期的数目取决于机器周期内完成微操作数目的多少以及相应功能部件的速度。一个机器周期的基本时钟周期数确定之后，还可以不断插入等待时钟周期，如图 7-16 所示。例如 PC 系列微机的一个总线周期（即机器周期）中包含了 4 个基本时钟周期 $T_1 \sim T_4$，在 T_3 和 T_4 之间可以插入若干个等待时钟周期 T_W 以等待速度较慢的存储部件或

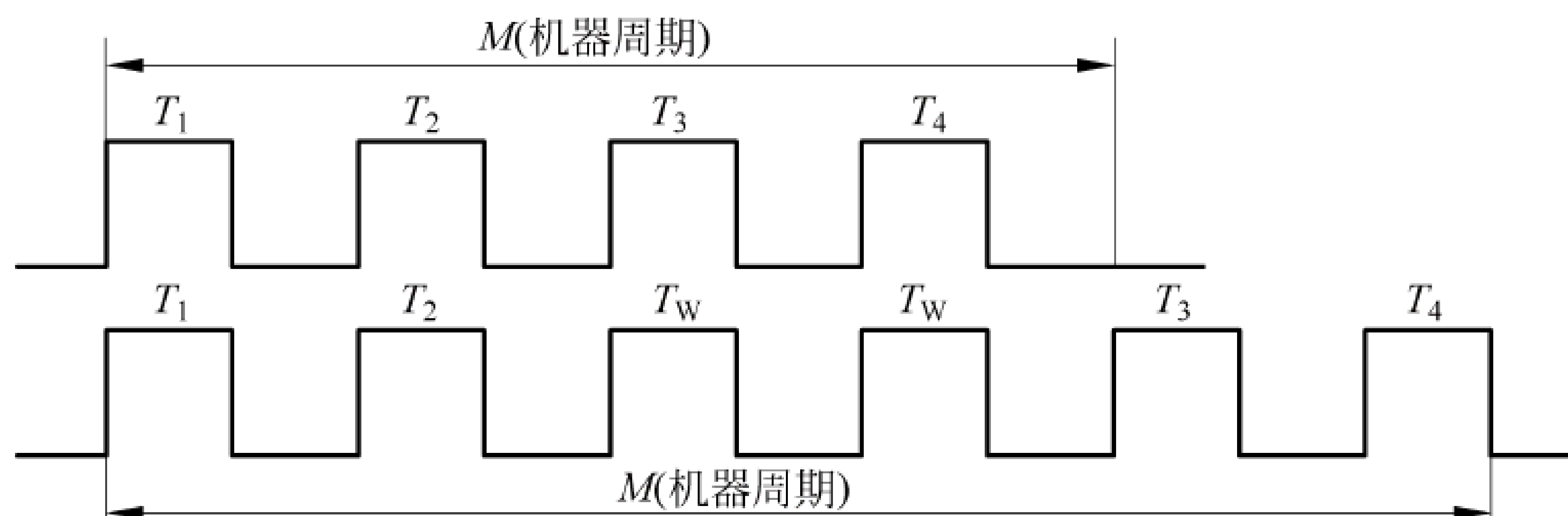


图 7-16　时钟周期的插入

I/O 设备完成读、写操作。

2. 工作脉冲

通常,在一个节拍内还设置一个或几个具有一定宽度的工作脉冲,以确保触发器可靠、稳定地翻转。在节拍中执行的微操作,有些需要同步定时脉冲,如将稳定的运算结果打入寄存器、机器周期状态切换等。

一般小型机中常采用如图 7-17 所示的周期-节拍-工作脉冲三级时序系统,图 7-17 中每个机器周期 M 中包括 4 个节拍 $T_1 \sim T_4$,每个节拍内有一个工作脉冲 P。

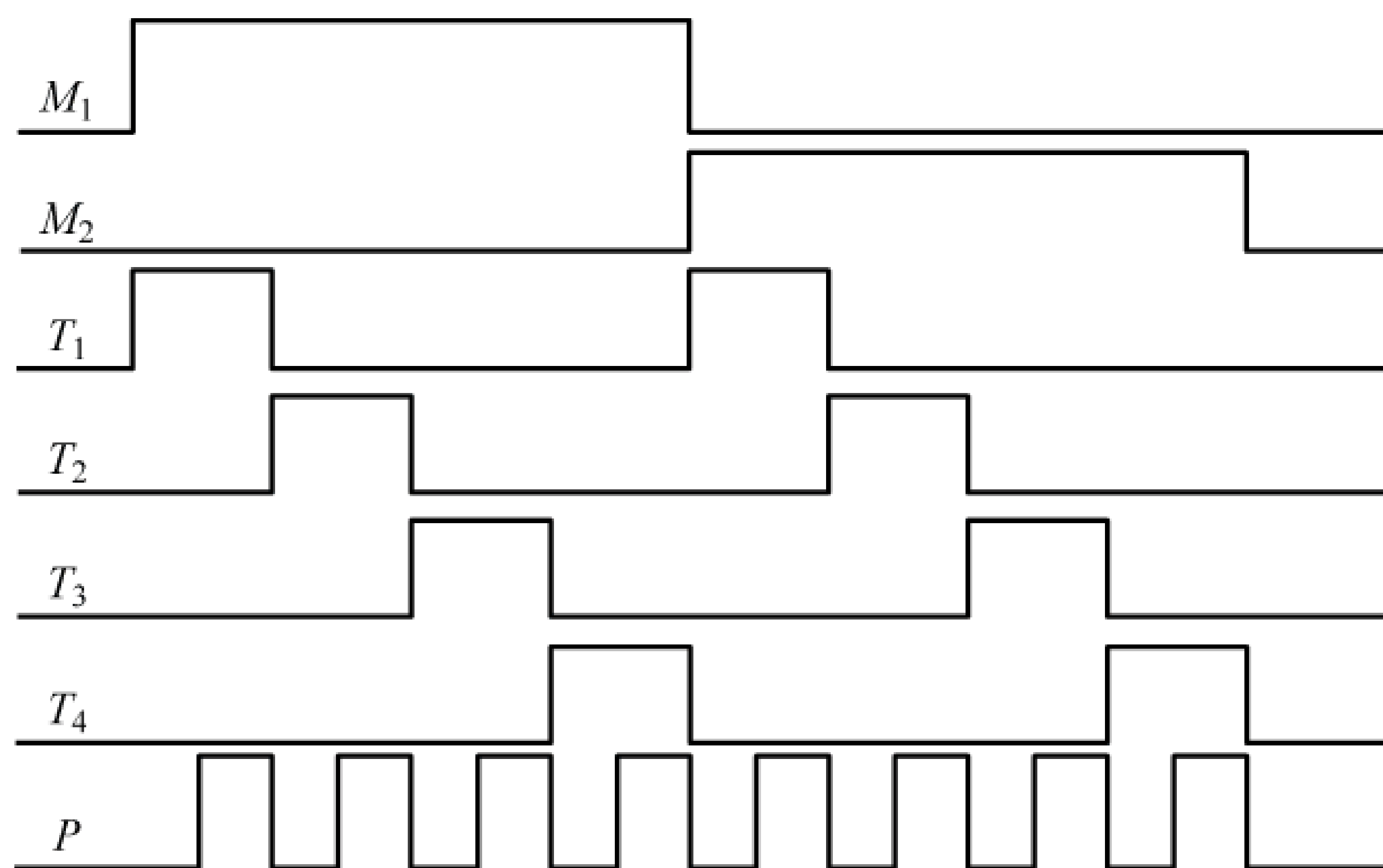


图 7-17 周期-节拍-工作脉冲三级时序系统

3. 节拍脉冲和工作脉冲的时间配合

在时序系统中,节拍脉冲和工作脉冲所起的作用是不同的,节拍脉冲信号在数据通路传输中起开门、关门的控制作用,是信息的载体。而工作脉冲则作为打入脉冲,作用在触发器的时钟脉冲输入端,起定时触发作用。通常触发器采用**节拍-脉冲**工作方法,例如对于 D 触发器,则节拍脉冲控制信息送到触发器的 D 端,工作脉冲送到 CP 端,它们之间的配合关系如图 7-18 所示。

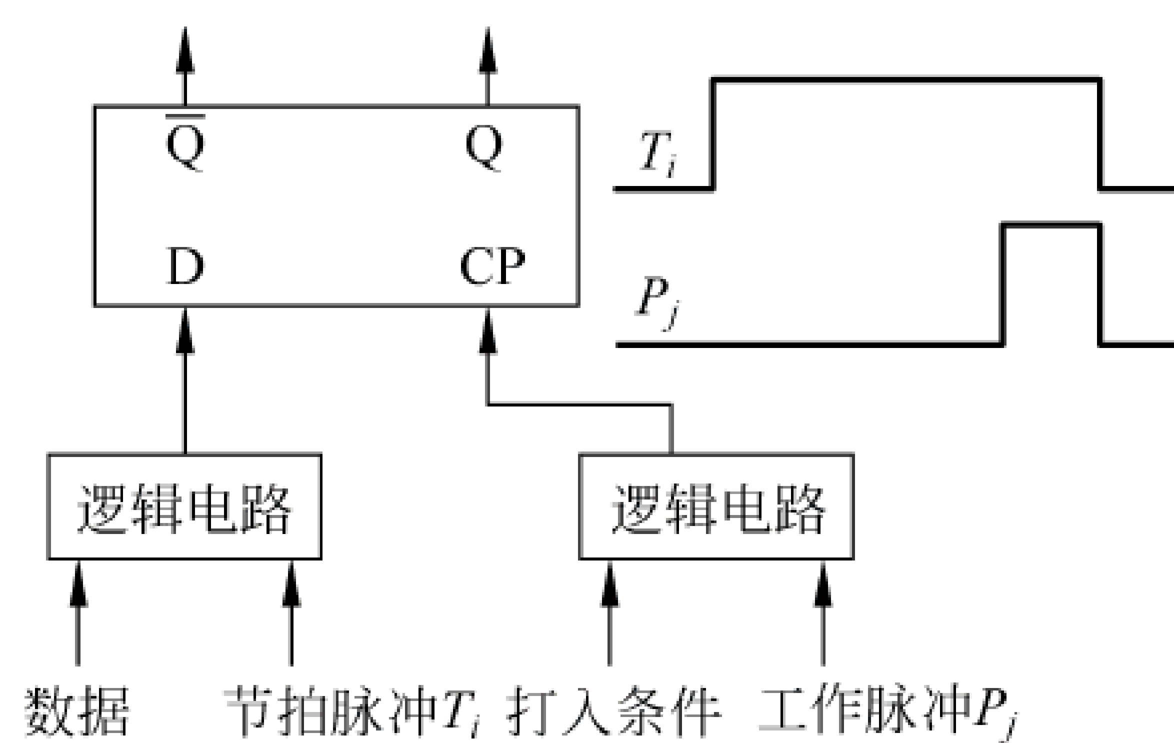


图 7-18 节拍脉冲和工作脉冲的配合关系

7.3.2 CPU 的控制方式

一条指令的执行是由许多微操作组成的,不同的指令对应着不同的微操作序列。控制器控制指令的执行过程是依次执行一个确定的微操作序列的过程。由于不同的机器指令所对应的微操作序列的长短和繁简程度各不相同,所以执行每条指令所需的时间也不相等,这就存在着用何种时序方式来形成微操作序列的问题。形成微操作序列的时序方式称为控制方式,它不仅直接决定着微操作控制信号的产生,也影响到控制器和其他部件的组成以及指令的执行速度等。常用的控制方式有同步控制方式、异步控制方式和联合控制方式等。

1. 同步控制方式

所谓同步控制方式是指系统有一个统一的时钟，所有的控制信号均来自这个统一的时钟信号。同步控制方式亦称为固定时序控制方式，其基本思想是选取部件中最长的操作时间作为统一的时间间隔标准，使所有的部件都在这个时间间隔内启动并完成操作。通常采用同步的时序发生器，产生固定的周而复始的机器周期电位、节拍脉冲，用统一的时序信号，定时各种操作，实现同步控制。

根据指令周期、机器周期和节拍的长度固定与否，同步控制方式又可分为以下几种。

(1) 定长指令周期：即所有指令的执行时间都相等。若指令的繁简差异很大，规定统一的指令周期，无疑会造成太多的时间浪费。因此，定长指令周期的方式很少被采用。

(2) 定长机器周期：这种方式中各个机器周期都相等，一般都等于主存的存取周期。而指令周期不固定，等于机器周期的整数倍。

(3) 变长机器周期、定长节拍：这种方式的指令周期长度不固定，而且机器周期长度也不固定，其含有的节拍数根据需要而定，与主存存取周期和总线周期没有固定关系。这种控制方式根据指令的具体要求和执行步骤，确定安排哪几个机器周期，以及每个机器周期中安排多少个节拍和工作脉冲，因此不会造成时间浪费；但时序系统的控制比较复杂，要根据不同情况确定每个机器周期的节拍数。

CPU 内部操作均采用同步控制，其原因是同一芯片的材料相同、工作速度相同，片内传输线短，又有共同的脉冲源，所以采用同步控制方式。

同步控制方式中，各条指令所需的时序由控制器统一发出，所有微操作都与时钟同步，所以又称为集中控制方式或中央控制方式。同步控制方式的优点是时序关系比较简单，控制器设计方便。但对简单的指令会产生空闲时间，降低了指令的执行速度。

2. 异步控制方式

异步控制方式即可变时序控制方式，各项操作不采用统一的时序信号控制，而根据指令或部件的具体情况决定，需要多少时间，就占用多少时间。

如图 7-19 所示的是一种采用“应答通信”实现的控制方式，各操作之间的衔接是由“结束-启动”信号来实现的。由前一项操作已经完成的“结束”信号，或由“准备好”信号来作为下一项操作的启动信号，在未收到“结束”或“准备好”信号之前不开始新的操作。例如，存储器读操作时，CPU 向存储器发一个读命令(启动信号)，启动存储器内部的时序信号，以控制存储器读操作，此时 CPU 处于等待状态。当存储器操作结束后，存储器向 CPU 发出 MFC (结束信号)，以此作为下一项操作的起始信号。

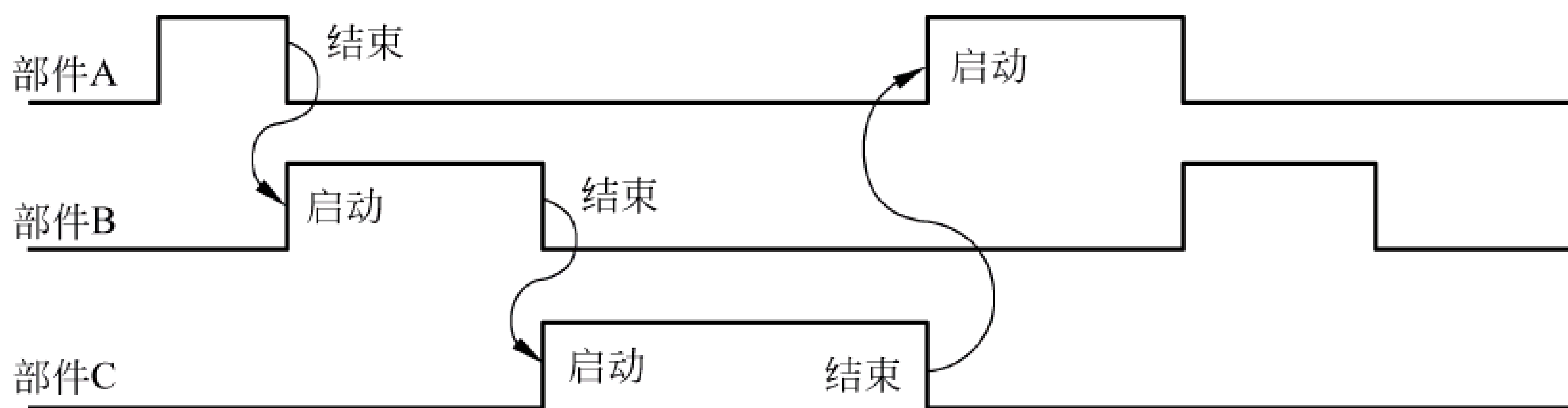


图 7-19　采用“应答通信”实现的控制方式

异步控制方式采用不同时序，没有时间上的浪费，因而提高了机器的效率，但是控制比较复杂。由于这种控制方式没有统一的时钟，而是由各功能部件本身产生各自的时序信号进行自我控制的，故又称为分散控制方式或局部控制方式。

3. 联合控制方式

CPU 内部的操作采用同步方式，CPU 与主存和 I/O 设备的操作采用异步方式，这就带来了一个同步方式与异步方式如何过渡、衔接的问题。也就是说，当主存或 I/O 设备的 Ready 信号到达 CPU 时，不可能恰好为 CPU 脉冲源的整周期或节拍的整周期。解决方法是一种折中方案，即联合控制方式。

联合控制方式是同步控制和异步控制相结合的方式。现代计算机中几乎没有完全采用同步或异步的控制方式，多数是采用联合控制方式。通常的设计思想是在功能部件内部采用同步方式或以同步方式为主的控制方式，在功能部件之间采用异步方式。

联合控制方式是介于同步和异步之间的一种折中。对大多数需要节拍数相近的指令，用相同的节拍数来完成，即采用同步控制；而对少数需要节拍数多的指令或节拍数不固定的指令，给予必要的延长，即采用异步控制。联合控制方式中，CPU 内部基本时序采用同步方式，当 CPU 通过总线与主存或其他 I/O 设备交换数据时，就转入异步方式。CPU 只需给出启动信号，主存和 I/O 设备按自己的时序信号去安排操作，一旦操作结束，则向 CPU 发送结束信号，以便 CPU 再安排它的后继工作。CPU 并不是在任何时刻就能立即对来自主存和 I/O 接口的请求信号做出反应的，而是在一个节拍周期的结束（下一个节拍周期的开始）。也就是说，当 CPU 进行主存的读写操作或进行 I/O 设备的数据传输时，是按同步方式插入一个节拍或几个节拍，直到主存或 I/O 设备的结束信号到达为止。联合控制方式是 CPU 进行主存的读写操作和 I/O 数据传输操作时通常采用的方式，能较好地解决同步与异步的衔接问题。

7.4 控制部件的硬布线实现

通过以上几节内容的介绍，已建立起了整机的概念。控制器是计算机中发布命令的“决策机构”，由程序计数器（PC）、指令寄存器（IR）、指令译码器、地址译码器、时序发生器和操作控制器组成。操作控制器根据指令操作码和时序信号，产生各种微操作控制信号，用以控制计算机各部分的操作，它是整个控制器的核心。

硬布线控制器是早期计算机的一种设计方法。这种方法是将控制部件做成产生固定时序控制信号的逻辑电路，该逻辑电路是由门电路和触发器构成的复杂树状网络，所以称为硬布线控制器，由于微操作控制信号序列是硬布线控制器中的组合逻辑网络产生的，因而又称为组合**逻辑控制器**。随着大规模集成电路制造技术的发展，使用通用可编程逻辑器件设计控制器，这种方式实现的控制器称为门阵列控制器；微操作控制信号序列也可用微程序产生，用这种方式实现的控制器称为微程序控制器，这部分内容将在 7.5 节介绍。

对实际使用的计算机，其控制器的组成是很复杂的。在本节介绍硬布线控制器设计时，采取了一种较为简单的机器组成结构，略去了许多烦琐的细节，重点突出控制器的设计思路及要点，以加深读者对整机概念及其工作原理的认识与掌握。

7.4.1 硬布线控制器的基本原理

硬布线控制器主要由组合逻辑网络、指令寄存器、指令译码器和节拍电位/节拍脉冲发生器等部分组成，由门电路和触发器构成的复杂树状网络实现，它以使用最少元件和取得最高操作速度为设计目标。

硬布线控制器的结构方框图如图 7-20 所示，其中组合逻辑网络产生计算机所需的全部微操作命令，是控制器的核心。

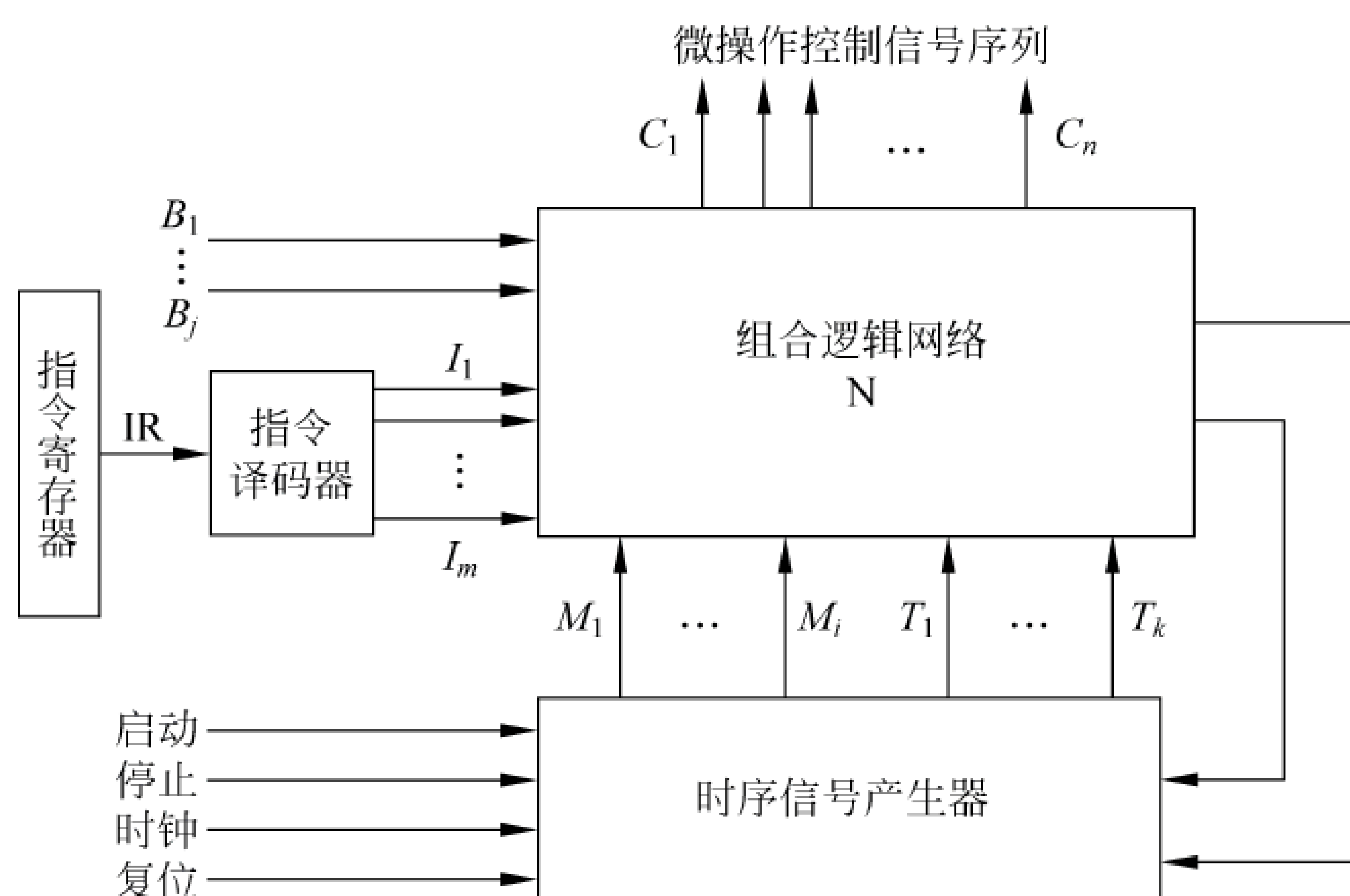


图 7-20 硬布线控制器结构方框图

组合逻辑网络的输入信号来源有三个：

(1) 来自指令操作码译码器的输出 $I_1 \sim I_m$，译码器每根输出线表示一条指令，译码器的输出反映出当前正在执行的指令；

(2) 来自执行部件的反馈信息 $B_1 \sim B_j$；

(3) 来自时序产生器的时序信号，包括节拍电位信号 $M_1 \sim M_i$ 和节拍脉冲信号 $T_1 \sim T_k$。其中节拍电位信号是机器周期(CPU 周期)信号，节拍脉冲信号是时钟周期信号。组合逻辑网络 N 的输出信号就是微操作控制信号 $C_1 \sim C_n$，用来对执行部件进行控制。另有一些信号则根据条件变量来改变时序发生器的计数顺序，以便跳过某些状态，从而可以缩短指令周期。

硬布线控制器的基本原理，归纳起来可叙述为：某一微操作控制信号 C_n 是指令操作码译码器输出 I_m，时序信号(节拍电位 M_i，节拍脉冲 T_k)和状态条件信号 B_j 的逻辑函数，其数学描述为

$$C_n = f(I_m, M_i, T_k, B_j)$$

控制信号 C_n 是用门电路、触发器等许多器件采用布尔代数方法来设计实现的。当机器加电工作时，某一操作控制信号 C_n 在某条特定指令和状态条件下，在某一操作的特定节拍电位和节拍脉冲时间间隔中起作用，从而激活这条控制信号线，对执行部件实施控制。显然，从指令流程图出发，就可以一个不漏地确定在指令周期中各个时刻必须激活的所有操作控制信号。例如，对引起一次主存读操作的控制信号 C_3 来说，当节拍电位 $M_1=1$，取指令时

被激活；而节拍电位 $M_4=1$，三条指令（LDA、ADD、SUB）取操作数时也被激活，此时指令译码器的 LDA，ADD、SUB 输出均为 1，因此 C_3 的逻辑表达式可由下式确定：

$$C_3 = M_1 + M_4(\text{LDA} + \text{ADD} + \text{SUB})$$

一般来说，还要考虑节拍脉冲和状态条件的约束，所以每一控制信号 C_n 可以由以下形式的布尔代数表达式来确定：

$$C_n = \sum_i \left(M_i \cdot T_k \cdot B_j \cdot \sum_m I_m \right)$$

7.4.2 硬布线控制器设计举例

一台实际机器的控制部件是非常复杂的，是有几百条输出控制线的逻辑网络，不可能在书中作详细的描述与设计。以最简单的模型机为实例，来讨论其设计的方法和步骤。

在这里仍以图 7-1 作为模型机。图中包含了各种部件设置以及它们之间的数据通路结构。在数据通路结构的基础上，就可拟出各种信息传送路径，以及为实现这些传送所需的微操作控制信号。

假设模型机的指令系统只包括 4 条典型的指令，如表 7-2 所示。

表 7-2 模型机的指令系统

助 记 符	操作描述与功能说明
LDA n	AC←n，把立即数 n 传送给累加寄存器 AC
ADD [X]	AC←(AC)+(X)，AC 的内容与存储单元 X 的内容相加送 AC
STR [X]	X←(AC)，AC 的内容存入存储单元 X
JMP X	PC←X，实现转移

(1) 根据 CPU 数据通路和指令功能，排列出每条指令的微操作控制序列。把表 7-2 的 4 条典型指令加以归纳，画出如图 7-21 所示的指令流程。

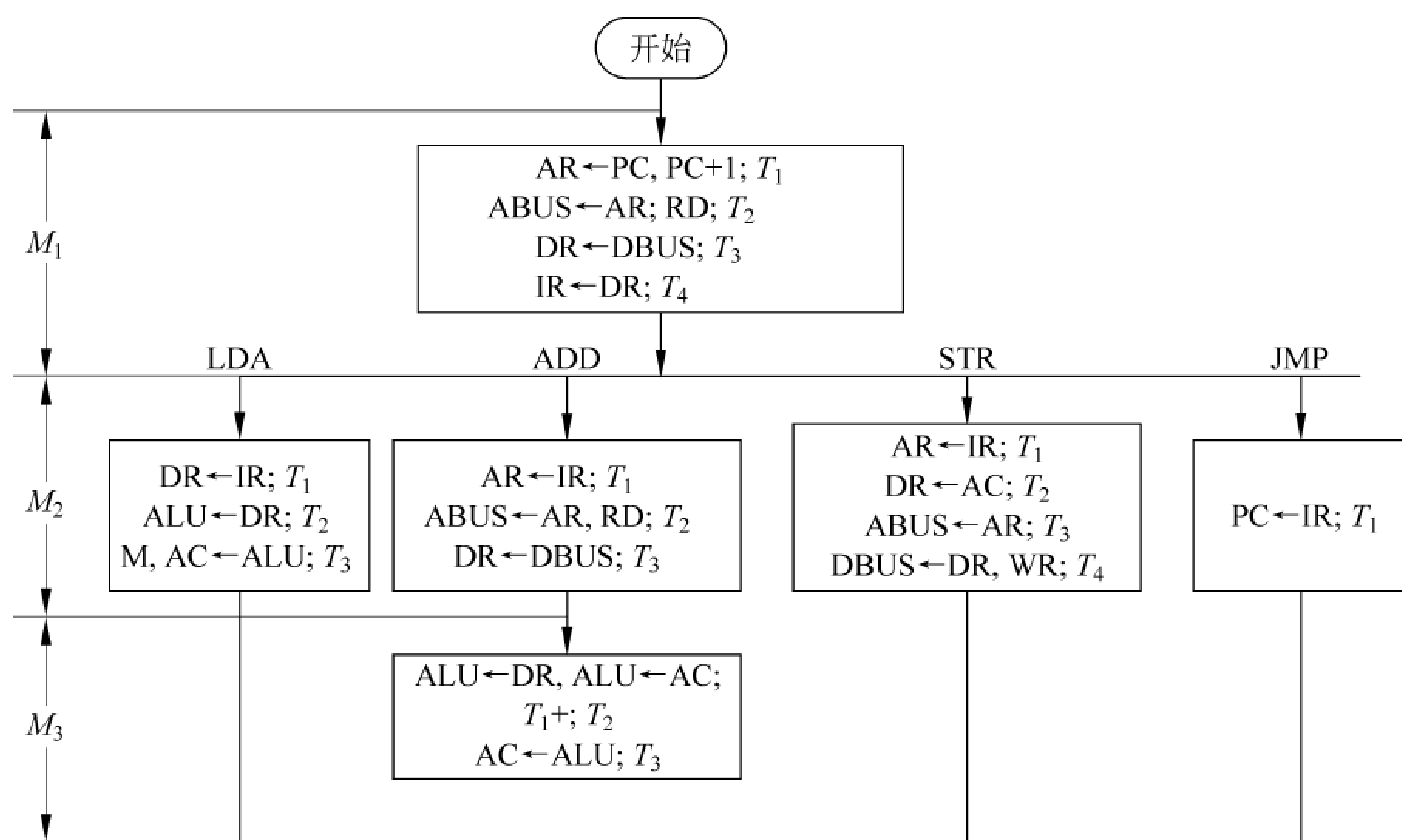


图 7-21 指令流程及节拍分配

由图 7-21 可知，所有指令的取指阶段都放在 M_1 节拍。在此节拍中，操作控制器发出微操作控制信号，完成从内存取出一条机器指令。

指令的执行阶段由 M_2、M_3 两个节拍完成。LDA、STR 和 JMP 指令只要一个节拍(M_2)即可完成，ADD 指令需要两个节拍(M_2,M_3)。为了简化节拍控制，指令的执行过程可采用同步控制方式，即各条指令的执行阶段均用最长节拍数 M_3 来考虑。这样，对 LDA、STR 和 JMP 指令来说，在 M_3 节拍没有什么操作。因此，同步控制方式虽然简化了节拍控制，但是，对于短指令来说，造成了时间上的浪费，降低了 CPU 的指令执行速度。在实际的机器中，在设计短指令流程时可以跳过某些节拍。当然在这种情况下，节拍信号发生器的电路要复杂一些。

(2) 微操作控制信号逻辑条件的综合与化简。

根据第(1)步列出的全部操作时间表，可列出全机在各种工作状态下所需的所有微操作控制信号，对于可以合并的微操作控制信号予以合并与简化。为了化简，常将周期电位、节拍等时间条件，以及操作码、寻址方式、寄存器号等逻辑条件综合为一些中间逻辑变量供使用。组合逻辑电路的输出是所需的微操作控制信号，可分为维持一个时钟周期宽的电位型微操作控制信号，或是短暂的、靠边沿作用的脉冲型微操作控制信号。

设每一个节拍电位中又由 4 个节拍脉冲组成，分别为 T_1、T_2、T_3、T_4，下面列出模型机微操作控制信号的逻辑表达式并加以简化：

$\mathrm{AR}\leftarrow\mathrm{PC}=M_1\cdot T_1$；

$\mathrm{PC}+1=M_1\cdot T_1$；

$\mathrm{ABUS}\leftarrow\mathrm{AR}=M_1\cdot T_2+M_2(\mathrm{ADD}\cdot T_2+\mathrm{STR}\cdot T_3)$；

$\mathrm{RD}=(M_1+\mathrm{ADD}\cdot M_2)\ T_2$；

$\mathrm{DR}\leftarrow\mathrm{DBUS}=(M_1+\mathrm{ADD}\cdot M_2)\ T_3$；

$\mathrm{IR}\leftarrow\mathrm{DR}=M_1\cdot T_4$；

$\mathrm{DR}\leftarrow\mathrm{IR}=\mathrm{LDA}\cdot M_2\cdot T_1$；

$\mathrm{AR}\leftarrow\mathrm{IR}=(\mathrm{ADD}+\mathrm{STR})\cdot M_2\cdot T_1$；

$\mathrm{PC}\leftarrow\mathrm{IR}=\mathrm{JMP}\cdot M_2\cdot T_1$；

$\mathrm{ALU}\leftarrow\mathrm{DR}=\mathrm{LDA}\cdot M_2\cdot T_2+\mathrm{ADD}\cdot M_3\cdot T_1$；

$\mathrm{DR}\leftarrow\mathrm{AC}=\mathrm{STR}\cdot M_2\cdot T_2$；

$M=\mathrm{LDA}\cdot M_2\cdot T_3$；

$\mathrm{AC}\leftarrow\mathrm{ALU}=(\mathrm{LDA}\cdot M_2+\mathrm{ADD}\cdot M_3)T_3$

$\mathrm{DBUS}\leftarrow\mathrm{DR}=\mathrm{STR}\cdot M_2\cdot T_4$；

$\mathrm{WR}=\mathrm{STR}\cdot M_2\cdot T_4$；

$\mathrm{ALU}\leftarrow\mathrm{AC}=\mathrm{ADD}\cdot M_3\cdot T_1$；

$+=\mathrm{ADD}\cdot M_3\cdot T_2$；

要说明的是：在上述逻辑表达式中，假定每个微操作控制信号都是持续一个节拍脉冲，即一个时钟周期的时间，这样简化问题。而实际的机器中各个微操作控制信号持续节拍脉冲数是不同的。

(3) 微操作控制信号的逻辑实现。

传统的方法是直接用门电路实现上述逻辑表达式，这一组电路就泛称为微操作控制信

号发生器，它们是控制器的主要实体。下面给出产生微操作控制信号 ABUS←AR 组合逻辑电路如图 7-22 所示[ABUS←AR＝ $M_1 \cdot T_2 + M_2(\text{ADD} \cdot T_2 + \text{STR} \cdot T_3)$]。

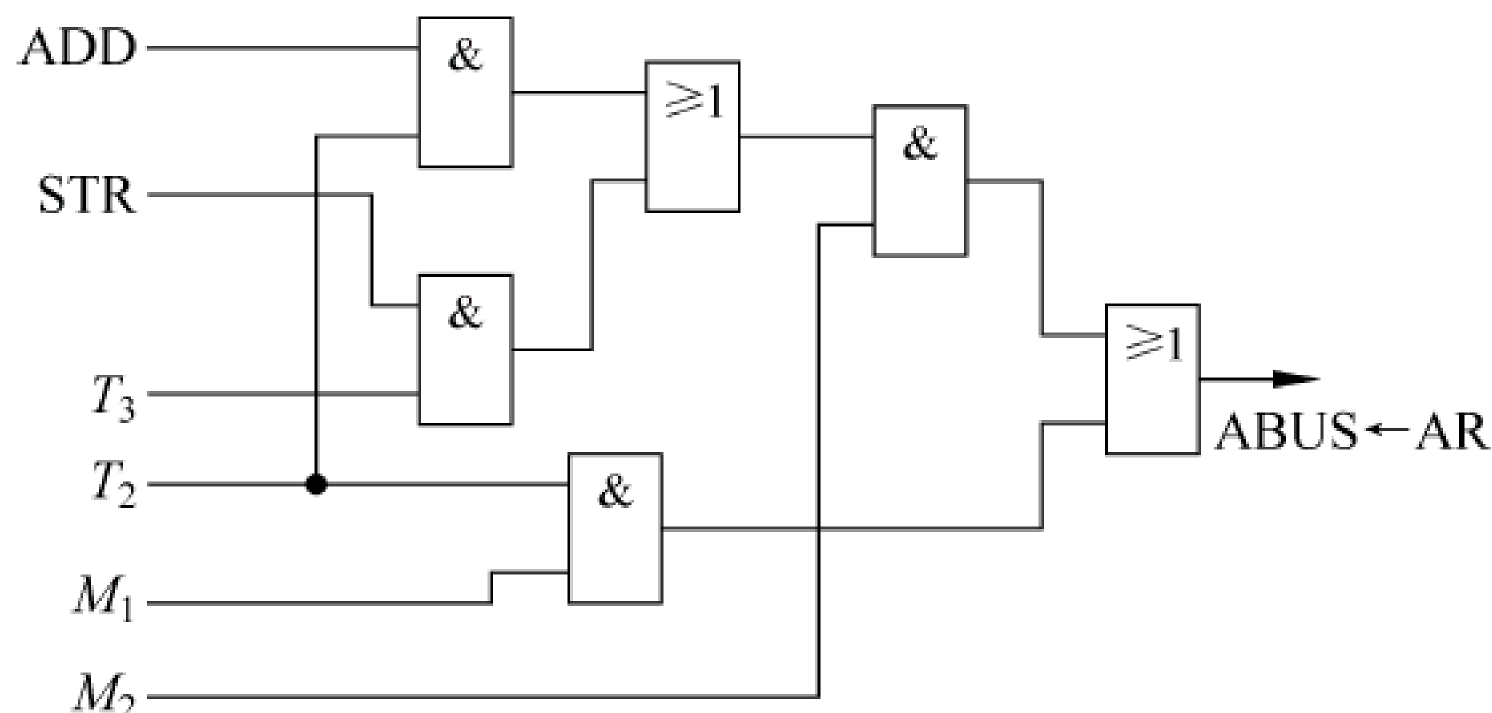


图 7-22 产生微操作控制信号 ABUS←AR 组合逻辑电路

图 7-22 中 M_1、M_2、T_2、T_3 是时序产生器的输出，ADD、STR 是指令译码器的输出，它们是组合逻辑网络的输入，产生的输出是微操作控制信号 ABUS←AR，其他的微操作控制信号也可以按类似的方法产生，在此不一一画出。产生所有微操作控制信号的逻辑电路构成了一个复杂的组合逻辑网络。一台计算机所需的微操作控制信号可能有成百上千，因此硬布线控制器中的微操作控制信号产生器，实际上是一组不规整的组合逻辑电路，即门电路集合。

随着大规模集成电路制造技术的发展，现在广泛采用可编程门阵列来实现控制器，或者尽量用门阵列产生大部分微操作控制信号，使电路结构得到简化。产生微操作控制信号的各种逻辑条件作为门阵列的输入，芯片内部产生相关的译码输出，再经编码形成若干微操作控制信号输出。这种多输入——多输出组合逻辑很适合构成微操作控制信号发生器。

(4) 硬布线控制器逻辑设计中的注意事项。

综上所述，硬布线控制器的设计是用大量的逻辑门电路，按一定规则组合成一个逻辑网络，从而产生各机器指令的微操作控制信号序列，其设计过程一般经历下列步骤：

① 根据给定的 CPU 数据通路和指令功能，排列出每条指令的微操作控制序列。

② 确定机器的状态周期、节拍与工作脉冲。根据指令的功能和器件的速度，确定指令执行过程中的周期及节拍的基本时间。

③ 列出每个微操作控制信号的逻辑表达式。确定了每条指令在每一状态周期中各个节拍内所完成的操作，也就得出了相应的操作控制信号的表达式。该表达式由指令操作码、时序状态(周期、节拍和工作脉冲)以及状态条件信息(允许有空缺)等因子组成。

④ 进行指令综合。综合所有指令的每一个操作命令，写出逻辑表达式，并进行化简。

⑤ 逻辑电路设计。合理选用逻辑门电路，将简化后的逻辑表达式用组合逻辑电路来实现。

总之，控制信号的设计与实现，技巧性较强，目前已有一些专门的开发系统或工具供逻辑设计使用，但是，对全局的考虑主要依靠设计人员的智慧和经验。

7.4.3 硬布线控制器的缺点及其改进

在比较完整地介绍了硬布线控制器的设计之后，将会发现硬布线控制方式的缺点，并找

到相应的改进方向。

(1) 如前所述,硬布线控制方式是用许多门电路产生微操作控制信号的,而这些门电路所需的逻辑形态很不规整,因此硬布线控制器的核心部分比较烦琐、零乱,设计效率较低,检查调试困难。就其设计方法而言,虽有一定的规律,但对于不同的指令与不同的安排,所构成的微操作控制信号形成的电路也就不同。改进的方向是将程序设计技术引入 CPU 机器的构成级,使设计规整化,也就是像编制程序那样去编制微操作控制信号序列。

(2) 硬布线控制方式的另一缺点是不易修改或扩展。这是因为设计结果用印刷电路板(硬布线)固定下来以后,就很难再修改与扩展了。而且,在各个微操作控制信号逻辑表达式中往往包含着许多条件,其中一些逻辑变量可能是其他微操作控制信号所公用的,修改一处就会牵动其他,因而很难改动。机器一旦设计并制造出来,要想修改其操作过程或某些操作的处理方式,或进一步修改与扩充指令系统,基本上是不可能的。这就使一台新设计的机器可能难以达到理想的效果;如果推出一种新的指令系统,原有的机器就不能执行它。改进的方向是将存储逻辑引入 CPU,取代组合逻辑的微操作控制信号发生器,也就是将微操作控制信号作为数字代码直接存入一个存储器中,只要修改所存储微操作控制信号的代码,就可修改有关的功能与执行的方式。

这两种改进方向导致了微程序控制思想的出现,但组合逻辑控制方式仍是产生控制命令的一种基本方式,即使采用微程序控制后,仍有部分微操作控制信号需要组合逻辑电路产生。组合逻辑控制方式的优点是速度较快,因此目前主要应用于高速计算机,如 RISC 处理器等。

7.5 微程序控制器

前面讲过,在计算机系统中,控制器常常采用硬布线设计技术和微程序设计技术,微程序控制设计技术是利用软件方法进行硬件设计的一门技术。本节将重点介绍微程序控制器。

早在 1951 年英国剑桥大学的 M. V. Wilkes 教授就提出了微程序控制的思想,他认为每条机器指令都可以分解为若干基本的操作序列,故一旦机器的体系结构和指令系统确定后,计算机所需要的全部操作控制信号也就确定了,可将这些操作信号以微码的形式编制成微指令,微指令中的每个二进制位定义成相应的控制码位,该位为 1,表示执行该操作,该位为 0,表示不执行该操作。这样,一条机器指令的执行可通过执行若干条微指令(这些微指令的集合叫做微程序)来实现。指令系统确定后,可将各条机器指令对应的微程序集中存放到一个只读存储器中(即控制存储器,简称控存),因此,控存实际上成了一个微操作控制信号发生器。

微程序控制器的基本设计思想概括起来有以下两点:

(1) 将指令周期中所需的微操作控制信号,以代码(微码)形式编成微指令,存入一个用 ROM 构成的控制存储器中。在 CPU 执行程序时,从控制存储器中取出微指令,其所包含的微命令控制有关操作。与硬布线控制方式不同,它由存储逻辑事先存储与提供微命令。这体现了 7.4 节提出的一种改进方向,将存储逻辑引入控制器的设计。

(2) 将各种机器指令的操作分解为若干微操作序列。每条微指令包含的微命令控制实

现一步操作，若干条微指令组成一段微程序，解释执行一条机器指令。针对整个指令系统的需要，编制出一套完整的微程序，事先存入控制存储器中。这体现了又一种改进方向，利用程序设计技术去编排指令的解释与执行。

上面从两个角度阐明了微程序控制的基本概念：微操作控制信号的产生方式，微程序与机器指令之间的对应关系。

上面所述涉及了两个层次，读者务必分清：

一个层次是程序员看到的传统机器级——机器指令。机器指令是提供给使用者编程的基本单位，它表明 CPU 所能完成的基本功能。程序员根据某项任务编制的工作程序，存放在主存储器中，最终成为可执行的指令序列，由程序计数器(PC)指示其流程。

另一个层次是设计者看到的微程序级——微指令。微指令是为实现机器指令中一步操作的微命令组合，它作为 CPU 内部的控制信息，通常不提供给使用者，对程序员是透明的。微程序是机器设计者事先编制好的，作为解释执行工作程序的一种硬件(固件)手段，在制造 CPU 时就存入控制存储器中。

工作程序可以很长，如几千条指令，但所使用的机器指令类型有限，所对应的微程序也是有限的。

7.5.1 微程序控制的基本概念

1. 微命令

一台数字计算机基本上可以划分为两大部分——控制部件和执行部件。控制器就是控制部件，而运算器、存储器、外围设备相对控制器来讲，就是执行部件。那么两者之间是怎样进行联系的呢?

控制部件与执行部件的一种联系是控制信息。控制部件通过控制线向执行部件发出各种控制命令，通常把这种控制命令叫做**微命令**，而执行部件接受微命令后所进行的操作就是**微操作**。

控制部件与执行部件之间的另一种联系是反馈信息。执行部件通过反馈线向控制部件反映操作情况，以便控制部件根据执行部件的“状态”下达新的微命令，这也叫做“状态测试”。

微操作在执行部件中是最基本的操作。由于数据通路的结构关系，微操作可分为相容性和相斥性两种。所谓**相容微操作**，是指同时或同一个微指令周期内可以并行执行的微操作。所谓**相斥微操作**，是指不能同时或不能在同一个微指令周期内并行执行的微操作。

为了说明微程序控制器的工作原理，将 CPU 简化为图 7-23。其中 ALU 为算术逻辑单元，AC 为累加寄存器，DR 为数据寄存器。AC、DR 寄存器的内容都可以送至 ALU 输入端，而 ALU 的输出可以送往 AC。在给定的数据通路中，每个控制门仅是一个常闭的开关，它的一个输入端代表来自寄存器的信息，而另一个输入端则作为操作控制端。一旦两个输入端都有输入信号时，它才产生一个输出信号，从而在控制信号能起作用的一个时间宽度中使信息在部件之间流动。

图 7-23 中每个开关由控制器中相应的微命令来控制。假定 ALU 只有加、减、传送三种操作，分别由 C_1、C_2、C_3 微命令来控制，这三种操作在同一个 CPU 周期中只能选择一种，不

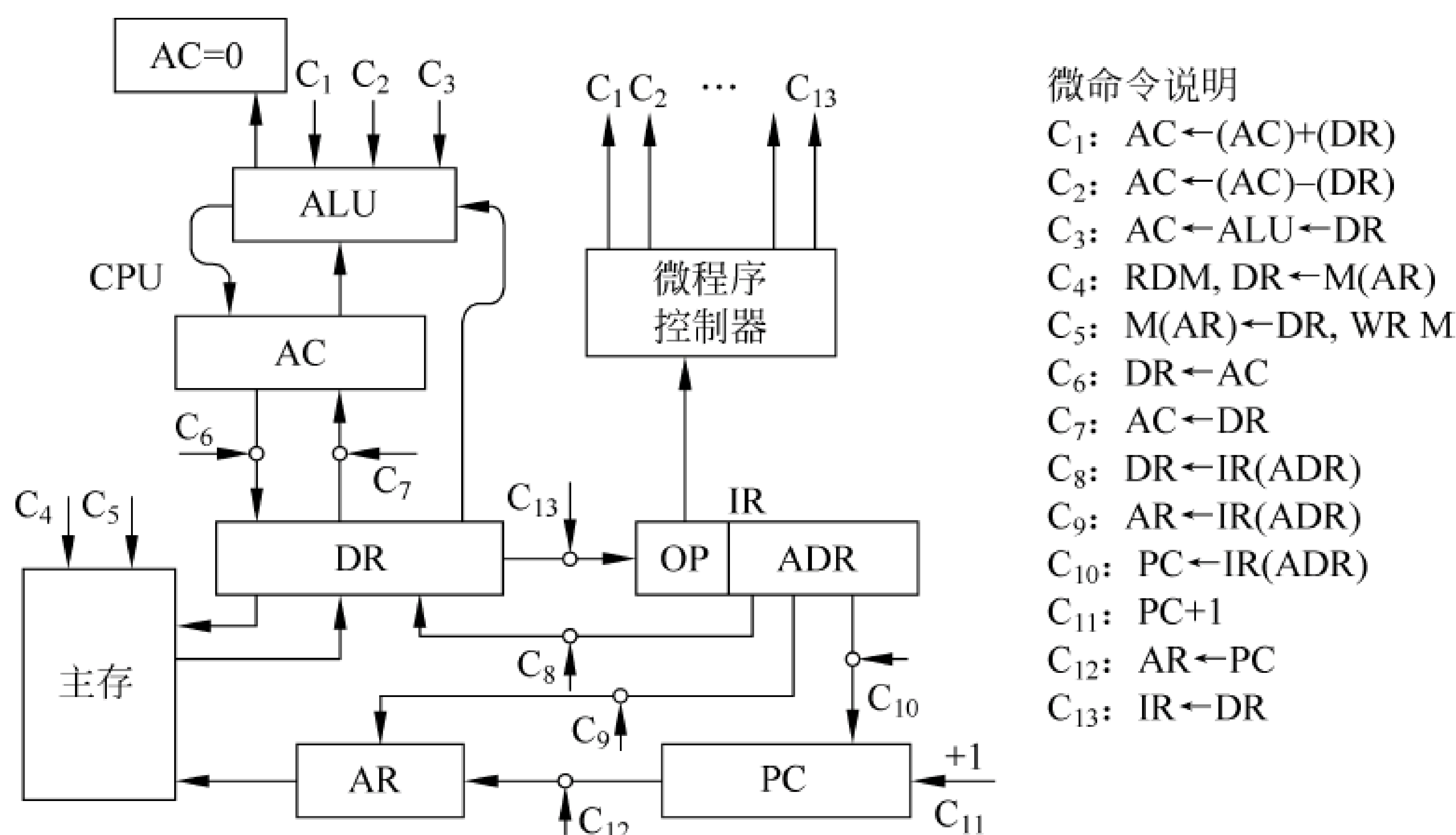


图 7-23 简化的 CPU

能并行，所以加、减、传送三个微操作是相斥性的微操作。AC＝0 为零标志触发器，运算结果为零时该触发器状态为“1”。类似地，C_4、C_5 分别是存储器读、写微命令，它们控制的存储器读、写微操作是相斥性的，C_6、C_7 控制的微操作也是相斥性。C_{11}、C_{12}、C_{13} 控制的微操作可以在同一个微指令周期中进行，所以是相容性的微操作。

2. 微指令和微程序

在机器的一个 CPU 周期中，一组实现一定操作功能的微命令的组合，构成一条**微指令**。以图 7-23 所示简化的 CPU 内的数据通路为例，图 7-24 表示一个具体的微指令结构，微指令字长为 19 位，它由操作控制字段和顺序控制字段两大部分组成。

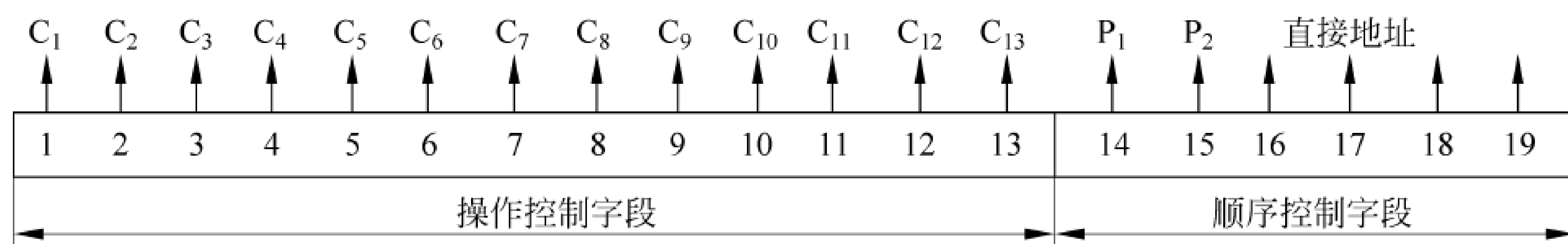


图 7-24 微指令的基本格式

1）操作控制字段

操作控制字段用来发出控制和指挥全机工作的控制信号。如图 7-24 所示的微指令中，该操作控制字段为 13 位，每一位表示一个微命令。每个微命令的编号同图 7-23 所示的数据通路相对应，具体功能说明在图 7-23 右半部分。当操作控制字段某一位信息为“1”时，表示发出相应的微命令；当某一位信息为“0”时，表示不发出该微命令。例如，当微指令字第一位信息为“1”时，表示发出 C_1 微命令，那么算术逻辑运算单元将执行 AC←(AC)＋(DR) 的微操作。同样，当微指令第二位信息为“1”时，则表示向 ALU 发出进行“－”的微命令，ALU 就执行减法操作。

微命令信号既不能来得太早，也不能来得太晚。因此，要求执行微命令信号时还要加入时间控制，即与时序信号相组合。

2）顺序控制字段

顺序控制字段用来决定产生下一条指令的地址。一条机器指令的功能是用多条微指令组成的序列来实现的，这些微指令序列通常叫做微程序。当执行当前一条微指令时，必须指出下一条微指令的地址，以便当前微指令执行完毕后，取出下一条微指令。决定下一条微指令地址的方法有多种，但基本上还是由微指令顺序控制字段的信息来决定的。

如图 7-24 所示的微指令中，顺序控制字段中的第 16～第 19 位将直接给出下一条微指令的地址，而第 14、第 15 两位作为判别测试标志。当第 14、第 15 两位为“0”时，表示不进行测试，直接按顺序控制字段第 16～第 19 位给出的地址取出下一条微指令；当第 14 位或第 15 位为“1”时，表示要进行 P_1 或 P_2 的判别测试，根据测试结果对第 16～第 19 位的某一位或几位进行修改，然后按修改后的地址取下一条微指令。

7.5.2 微程序控制器的组成

微程序控制器原理框图如图 7-25 所示。它主要由控制存储器(CM)、微指令寄存器(μIR)、微地址寄存器(μAR)和微地址形成逻辑 4 部分组成。

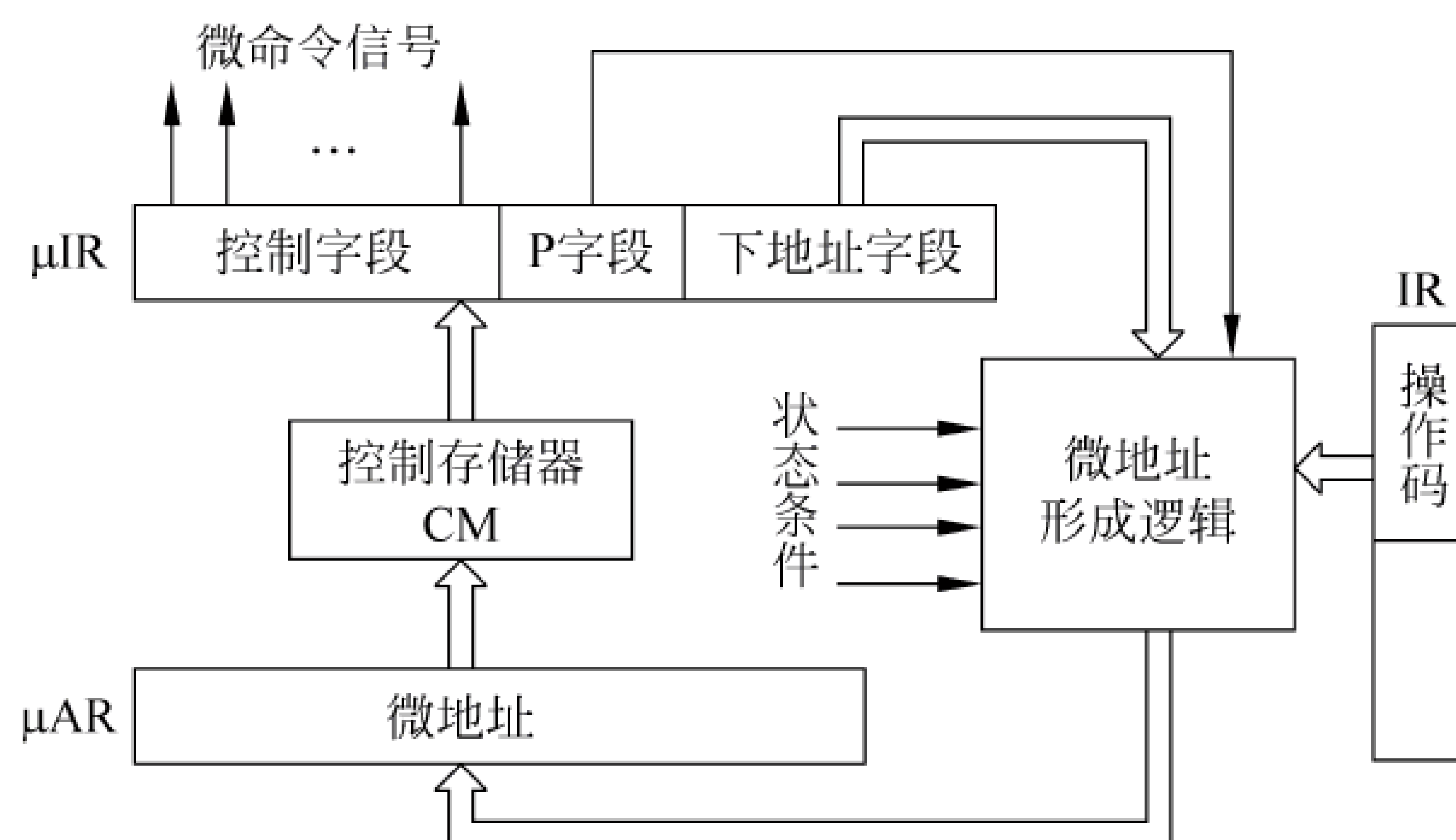


图 7-25 微程序控制器组成原理框图

1. 控制存储器

控制存储器(CM)用来存放实现全部指令系统的微程序。控制存储器是一种只读型存储器，微程序固化在其中，微指令只能读出而不能改写。其工作过程是：每读出一条微指令，则执行这条微指令；接着又读出下一条微指令，又执行这一条微指令……读出一条微指令并执行微指令的时间总和称为一个微指令周期。在串行方式的微程序控制器中，微指令周期就是只读存储器的工作周期。控制存储器的字长就是微指令字的长度，其存储容量取决于机器指令系统中指令的功能及数量。对控制存储器的要求是速度要快，读出周期要短，通常采用高速半导体存储器来构造。

2. 微指令寄存器

微指令寄存器(μIR)用来存放由控制存储器读出的一条微指令信息，包括操作控制字段、判别测试字段和下地址字段的信息。其中操作控制字段提供微命令信号，可以直接给出，也可以通过译码而提供，这取决于微指令的编码方式；而判别测试字段和下地址字段决

定将要访问的下一条微指令的地址。

3. 微地址寄存器

微地址寄存器(μAR)存放要访问的微指令在控制存储器的地址,该地址由微地址形成逻辑提供。

4. 微地址形成逻辑

在一般情况下,由控制存储器读出的微指令直接给出下一条微指令的地址,通常简称为微地址。如果微程序不出现分支,下一条微指令的地址就直接由微指令寄存器(μIR)的下地址字段决定。当微程序出现分支时,意味着微程序出现条件转移。在这种情况下,通过判别测试字段 P 和执行部件的"状态条件"反馈信息,去修改微地址寄存器的内容,并按改好的内容去读取下一条微指令。因此,微地址形成逻辑就承担自动完成修改地址的任务。

7.5.3 微程序设计举例

一条机器指令是由若干条微指令组成的序列来实现的。因此,一条机器指令对应着一个微程序,而微程序的总和便可实现整个指令系统。

现在以表 7-3 所示的模型机指令系统为例,具体看一看微程序控制的过程。

表 7-3 模型机指令系统

指令助记符	操作描述与功能说明
LDA n	AC←n,把立即数 n 传送给累加寄存器 AC
ADD [X]	AC←(AC)+(X),AC 的内容与存储单元 X 的内容相加送 AC
SUB [X]	AC←(AC)-(X),AC 的内容减存储单元 X 的内容送 AC
STR [X]	X←(AC),AC 的内容存入存储单元 X
JZ X	当 AC=0 时,PC←X; 否则,PC 不变

1. 分析机器指令的功能

模型机指令系统共有 5 种指令,采用 2 种寻址方式。

LDA n,数据传送指令,采用立即寻址方式,把立即数 n 传送给累加寄存器 AC,给寄存器赋初值。

ADD [X],加法指令,采用直接寻址方式,指令执行时,先根据有效地址 X 访问存储器取操作数,然后与累加寄存器 AC 的内容相加,结果送 AC。

SUB [X],减法指令,采用直接寻址方式,指令执行时,先根据有效地址 X 访问存储器取操作数,然后用 AC 的内容减去从存储器取的操作数,结果送 AC。

STR [X],存操作数指令,采用直接寻址方式,指令执行时,AC 的内容存入存储单元 X。

JZ X,条件转移指令,当 AC=0 时,条件满足,用 X 修改 PC 的内容,实现程序转移;否则,PC 不变,继续执行下一条指令。

2. 设计微程序流程图

机器指令要被执行,首先要完成取指令的操作。

第一条微指令是“取指”微指令，它是一条专门用来取机器指令的微指令，任务有三个：从内存取出一条机器指令，并把它存放到指令寄存器中。对程序计数器加 1，做好取下一条机器指令的准备；对机器指令的操作码用 P_1 进行判别测试，然后修改微地址寄存器的内容，给出下一道微指令的地址，转向相应指令的微程序段。

模型机指令系统的微程序流程图如图 7-26 所示。可以看到，每一条微指令用一个长方框表示。每一微指令的地址用数字标于长方框的右上角。注意，菱形框代表判别测试，它的动作依附于上一条微指令。当微程序转向公操作(用符号“⌐”表示)时，如果没有外围设备请求服务，那么将转向取下一条机器指令。

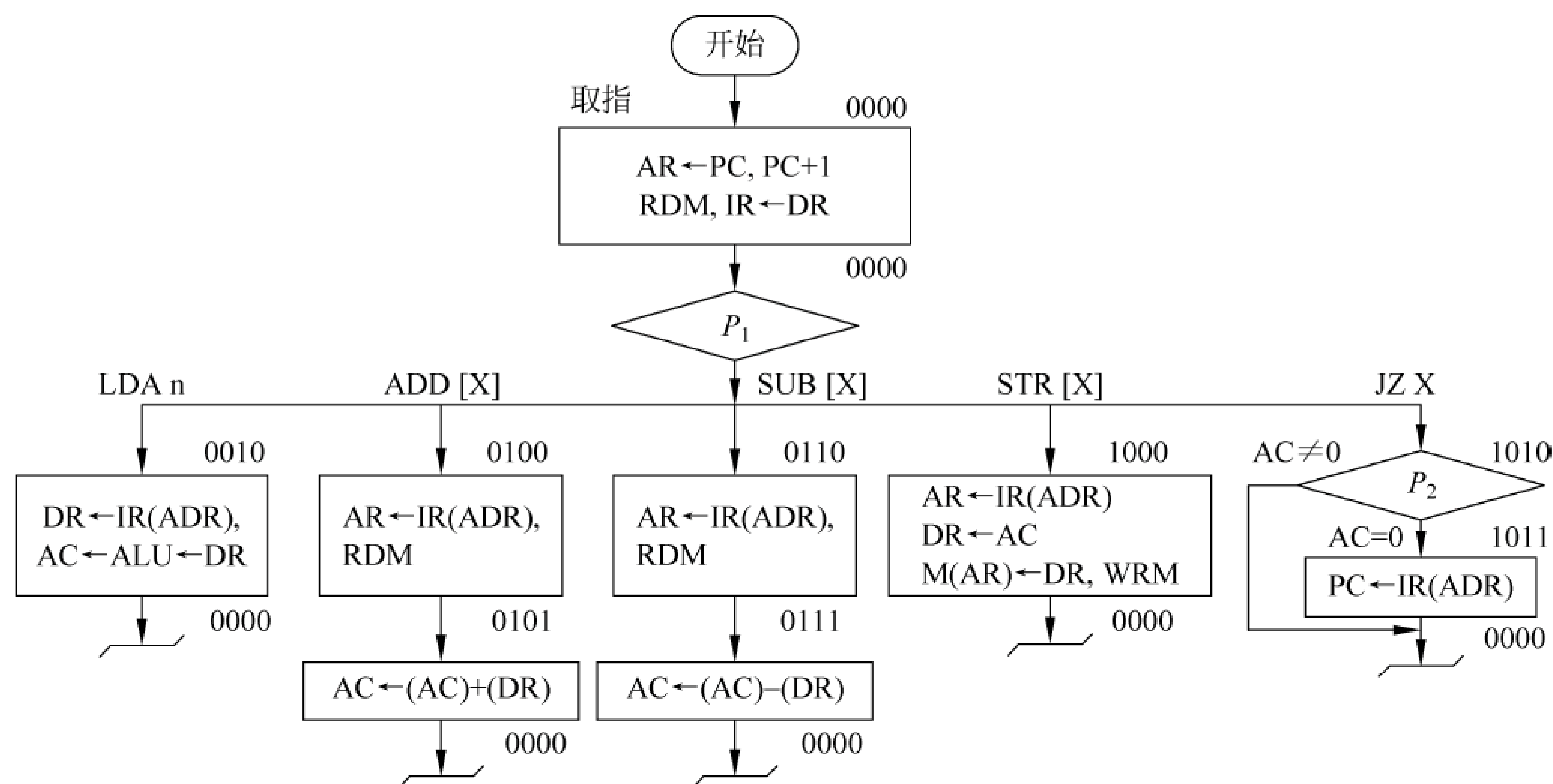


图 7-26　模型机指令系统的微程序流程图

3. 编写微程序

假定模型机的 CPU 如图 7-23 所示，采用如图 7-24 所示的微指令格式，模型机指令系统的微程序流程图如图 7-26 所示，这样，就可编写出如表 7-4 所示的模型机指令系统的微程序代码。

表 7-4　模型机指令系统的微程序代码

控存地址	控制字段 $C_1C_2C_3C_4\cdots C_{13}$	测试字段 P	后继微地址字段
0000	0001 0000 0011 1	1 0	0000
0010	0010 0001 0000 0	0 0	0000
0100	0001 0000 1000 0	0 0	0101
0101	1000 0000 0000 0	0 0	0000
0110	0001 0000 1000 0	0 0	0111
0111	0100 0000 0000 0	0 0	0000
1000	0000 1100 1000 0	0 0	0000
1010	0000 0000 0000 0	0 1	0000
1011	0000 0000 0100 0	0 0	0000

假设把已经编好的微程序存放到控制存储器中，当机器启动时，只要给出控制存储器的首地址，就可以调出所需要的微程序。

为此，首先给出第一条微指令的地址0000，经地址译码，控制存储器选中所对应的“取指”微指令，并将其读到微指令寄存器(μIR)中，第一条微指令的二进制编码如表7-4所示。它的操作控制字段要发出4个微命令：发出C_{12}微命令，将PC内容送到地址寄存器(AR)；发出C_{11}微命令，PC自加1；发出C_4微命令，内存执行读操作，从内存单元取出机器指令代码送到缓冲寄存器DR；发出C_{13}微命令，将DR中的机器指令代码再送到指令寄存器(IR)。C_{11}发出了PC+1微命令，使程序计数器加1，做好取下一条机器指令的准备。另一方面，微指令的顺序控制字段指明下一条微指令的地址是0000，但是由于判别字段中第14位为1，表明是P_1测试，因此0000不是下一条微指令真正的地址。P_1测试的“状态条件”是指令寄存器的操作码字段，即用OP字段作为形成下一条微指令的地址，于是微地址寄存器的内容修改成该机器指令微程序段在控存中的地址。

如果在第一个CPU周期完成后，取得的指令是JZ X，经P_1测试形成的下一条微指令的地址是1010，按照1010这个微地址读出第二条微指令，它的二进制编码如表7-4所示。在这条微指令中，顺序控制部分由于判别字段中P_2为1，表明进行P_2测试，测试的“状态条件”为进位标志AC=0。换句话说，此时微地址0000需要进行修改，当AC=0时，下一条微指令的地址为1011；当AC≠0时，下一条微指令的地址为0000。显然，在测试一个状态时，有两条微指令作为要执行的下一条微指令的“候选”微指令。现在假设AC=0，则要执行的下一条微指令地址为1011。按照1011这个微地址读出的微指令，其控制字段发出C_{10}微命令，完成修改PC的操作，从而实现了条件转移。

从这个简单的控制模型中，就可以看到微程序控制的主要思想及工作原理。

按上述步骤把指令系统中的所有指令的微程序段都编写好，然后固化在控存中，这样就完成了微程序控制器的设计。要注意的是，对于不同指令的微程序段的公操作用一样的微指令，这样可节省控存的容量。

下面介绍微程序控制的计算机的工作过程。计算机加电以后，首先由复位信号(Reset)将开机后执行的第一条指令的地址送入PC内，同时将一条“取指”微指令送入微指令寄存器中，并将其他一些有关的状态位或寄存器置于初始状态。当电压达到稳定值后，自动启动计算机，产生节拍电位和工作脉冲。为保证计算机正常工作，电路必须保证开机后第一个机器周期信号的完整性，在该CPU周期末，产生开机后的第一个工作脉冲。然后计算机开始执行程序，不断地取出指令、执行指令。

程序可以存放在固定存储器中，也可以利用固化在只读存储器(ROM)中的一小段引导程序，将要执行的程序和数据从外部设备调入主存。实现各条指令的微程序是存放在微程序控制器中的。当前正在执行的微指令从微程序控制器中取出后放在微指令寄存器中，由微指令的控制字段中的各位直接控制信息和数据的传送，并进行相应的处理。当遇到停机指令或外来停机命令后，应该等当前这条指令执行完后再停机或至少在本机器周期结束时停机。要保证停机后，重新启动计算机能继续工作而且不出现任何错误。

前面介绍了微程序控制计算机工作的过程。实现这一过程的方法有很多，硬件结构也有很大的差别，这里只是简单说明一种实现的方法。

7.5.4 微程序控制的特点

1. 微指令周期与 CPU 周期的关系

在串行方式的微程序控制器中，**微指令周期**等于读出微指令的时间加上执行该条微指令的时间。为了保证整个机器控制信号的同步，可以将一个微指令周期时间设计得恰好和 CPU 周期时间相等。图 7-27 给出了微指令周期与 CPU 周期的时间关系。

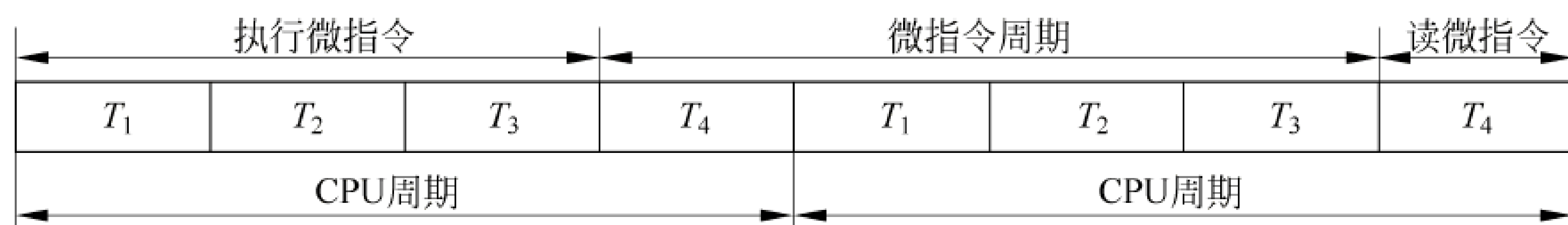


图 7-27 微指令周期与 CPU 周期的时间关系

一个 CPU 周期包含 4 个等间隔的节拍脉冲 $T_1 \sim T_4$，若每个脉冲宽度为 200ns。用 T_4 作为读取微指令的时间，用 $T_1+T_2+T_3$ 时间作为执行微指令的时间。

例如，在前 600ns 时间内运算器进行运算，在 600ns 时间的结束时运算器已经运算完毕，可用 T_4 上升沿将运算结果打入某个寄存器。同时可用 T_4 间隔读取下条微指令，经 200ns 的时间延迟，下条微指令又从控制存储器读出，并在 T_1 上升沿打入微指令寄存器。如忽略触发器的翻转延迟，那么下条微指令的微命令信号就从 T_1 上升沿开始有效，直到下一条微指令读出后打入微指令寄存器为止。因此一条微指令的保持时间恰好是 800ns，也就是一个 CPU 周期的时间。

2. 微指令与机器指令的关系

通过前面的学习，能全面了解微指令与机器指令的关系。现在把微指令与机器指令之间的关系归纳如下：

(1) 一条机器指令对应一段微程序，而微程序是由若干个微指令序列组成的，因此，一条机器指令的功能是由若干条指令组成的序列来实现的，即一条机器指令所完成的操作可通过若干条微指令来完成，由微指令进行解释和执行。

(2) 从指令与微指令、程序与微程序、主存与控存、地址与微地址的一一对应关系可知，指令、程序和地址与主存储器有关；而微指令、微程序和微地址与控制存储器有关，并且也有相对应的硬设备。微指令与机器指令间的关系如图 7-28 所示。

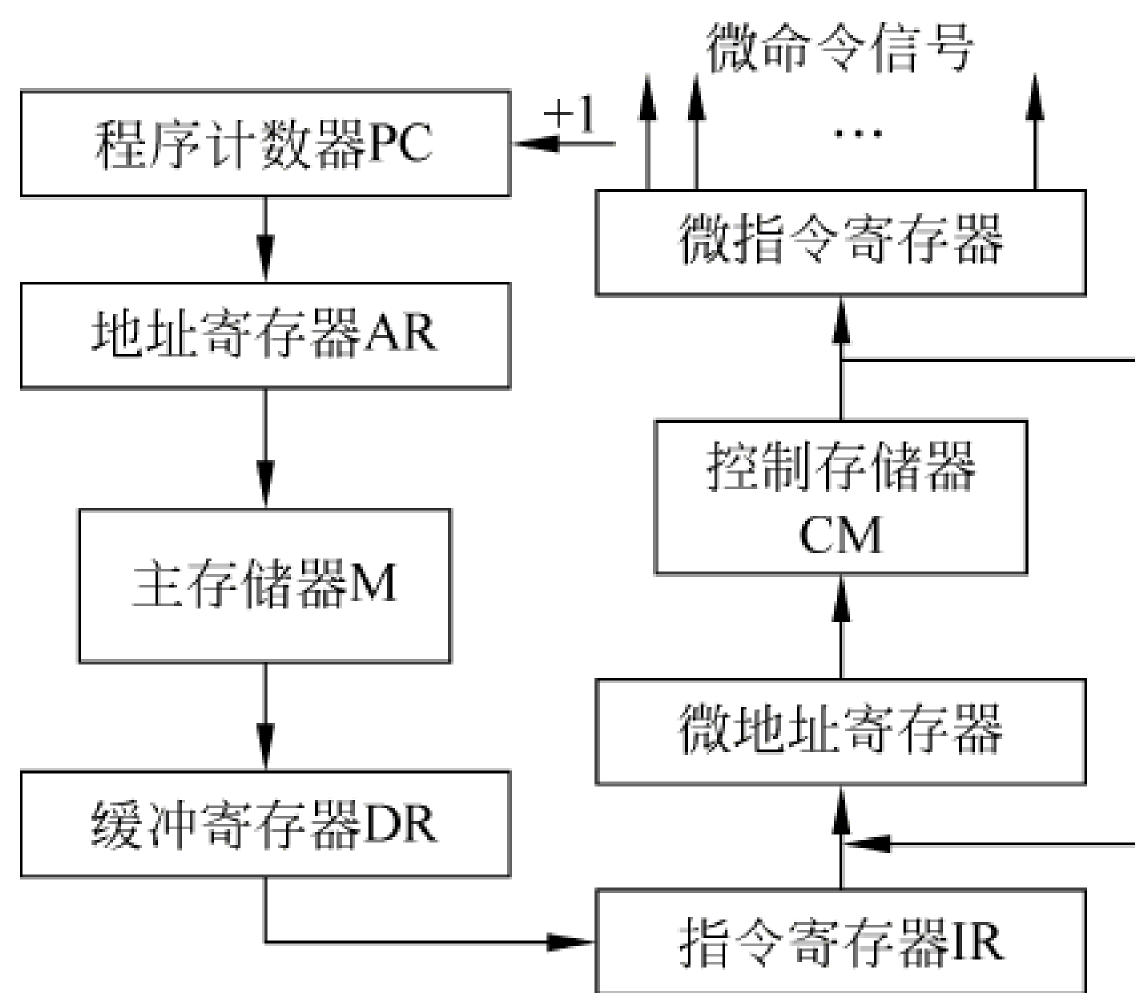


图 7-28 微指令与机器指令之间的关系图

(3) 由于一个 CPU 周期对应一条微指令，在 7.2 节曾讲述指令周期与机器周期的概念，并归纳了 4 条典型指令的指令周期。从指令的微程序流程图中看到设计微程序的流程，也可进一步明确机器指令与微指令的关系。

3. 微程序控制器与硬布线控制器的比较

硬布线控制器与微程序控制器相比较，在操作控制信号的形成上有较大的区别外，其他没有本质的区别。对于实现相同的一条指令，不管是采用硬布线控制还是采用微程序控制技术，都可以采用多种设计方案，导致了各种不同的控制器在具体实现方法和手段上的区别，性能也会有所差异。

微程序控制器与硬布线控制器的主要区别可归纳为以下两点。

1）实现方式

微程序控制器的控制功能是在存放微程序的控制存储器和存放当前正在执行的微指令的寄存器的直接控制下实现的，而硬布线控制器的控制功能则由逻辑门组合实现。微程序控制器的电路比较规整，各条指令信号的差别集中在控制存储器内容上，因此，无论是增加或修改指令都只要增加或修改控制存储器内容即可。若控制存储器是ROM，则要更换芯片，在设计阶段可以先用RAM或EPROM来实现，验证正确后或成批生产时，再用ROM代替。硬布线控制器的控制信号先用逻辑表达式列出，经化简后用电路来实现，因此，显得零乱复杂，当需要修改指令或增加指令时就必须重新设计电路，非常麻烦而且有时甚至无法改变。因此，微程序控制取代了硬布线控制并得到了广泛应用，尤其是指令复杂的计算机，一般都采用微程序来实现控制功能。

2）性能方面

在同样的半导体工艺条件下，微程序控制的速度比硬布线控制的速度慢，因为执行每条微指令都要从控制存储器中读取，影响了速度；而硬布线控制逻辑主要取决于电路延时，因而在超高速机器中，对影响速度的关键部分和（核心部件CPU）往往采用硬布线逻辑实现。近年来，在一些新型的计算机系统中，例如，RISC（精简指令系统计算机）中，一般均选用硬布线逻辑电路。

7.6 微程序设计技术

根据7.5节的内容我们知道，微程序就是微指令的集合。在实际设计中所追求的应是尽量减小控存的容量，提高微程序的执行效率，设计灵活，便于修改。而控存容量的大小、微程序的执行效率很大程度上取决于微指令的结构，因此，微程序设计的关键是如何确定微指令的结构，设计微指令结构应当追求的目标是：

（1）有利于缩短微指令字长度；

（2）有利于减小控制存储器的容量；

（3）有利于减少微程序的长度；

（4）有利于提高微程序的执行速度；

（5）有利于对微指令进行修改；

（6）有利于提高微程序设计的灵活性。

这些也是微程序设计技术所要讨论的问题。

7.6.1 微命令编码

前面已讲到，微指令是由操作控制字段和顺序控制字段所组成的。微命令编码，就是对微指令中的操作控制字段进行编码表示，通常有以下几种编码方法。

1. 直接表示法

采用直接表示法的微指令结构如图 7-29 所示，其特点是操作控制字段中的每一位代表一个微命令，在设计微指令时，只要将微指令操作控制字段相应的位置成“1”或“0”，便可发出或禁止某个微命令，这就是直接表示法。

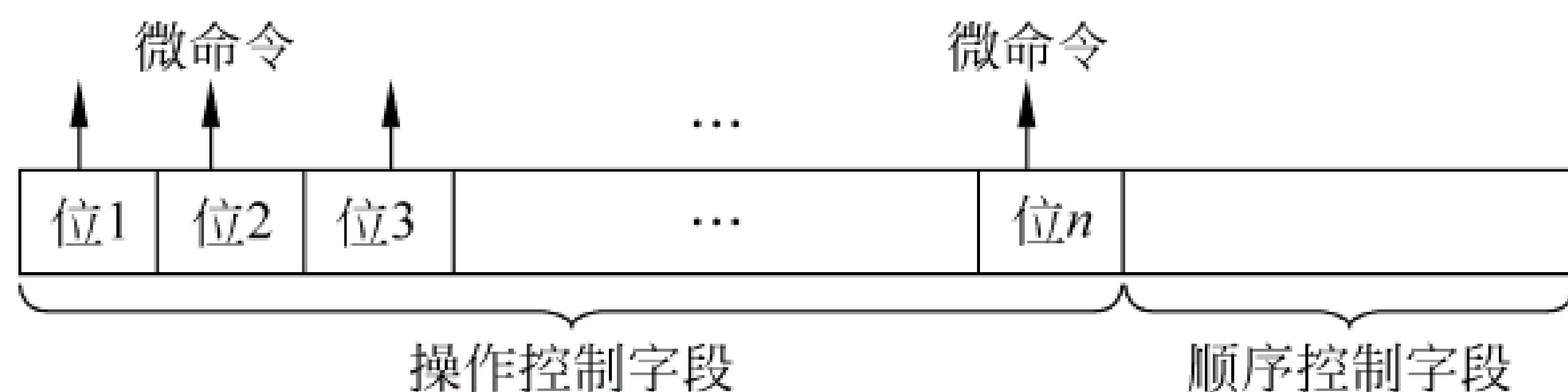


图 7-29 微指令结构——直接表示法

这种方法的优点是简单、直观，输出直接用于控制，微命令的并行控制能力强，编写的微程序行数少。缺点是微指令字较长，如果一台计算机的微命令有 400 个，则微指令的控制字段就需 400 位，这会使微指令字的长度达到难以接受的地步，而且，对如此长的微指令字，在给定的任何一条微指令中，常常仅有少数几个微命令，因此只有少数几位置 1，造成位空间不能充分利用，也造成了控存空间的浪费。因此，为了缩短微指令字的长度，可以采用字段直接译码法。

2. 字段直接译码法

事实上，在控制数据通路的操作中，大多数微命令不是同时需要，而且许多微命令是互相排斥。例如，控制 ALU 操作的各种微命令(如＋、－、Move 等)不能同时出现，即在一条微指令中只能出现一种运算操作；又如存储器的读/写操作也不能同时出现。通常将在同一个微指令周期中不能同时出现的微命令称为相斥性微命令，将在同一个微指令周期中可以同时出现的微命令称为相容性微命令。

字段直接译码法，是把一组相斥性的微命令组成一个字段，用字段的不同代码表示不同的微命令，然后通过字段译码器对每一个微命令进行译码，译码输出作为微操作控制信号，其微指令结构如图 7-30 所示。

字段直接译码法可以缩短微指令字的长度。例如，某机器指令系统总共需要 300 个微命令，采用直接表示法，微指令的操作控制字段需 300 位。采用字段直接译码法，如用 4 位二进制代码表示一个字段，经字段译码器输出，可产生 15 个相斥的微命令，总共 20 个字段就可以获得 300 个微命令。注意：通常全“0”编码表示不发出任何微命令。微指令的操作控制字段仅 80 位。在此虽然增加了一些译码线路，但使控制存储器的容量大大缩小，这是一种较为经济的做法。

字段直接译码法的分段原则如下：

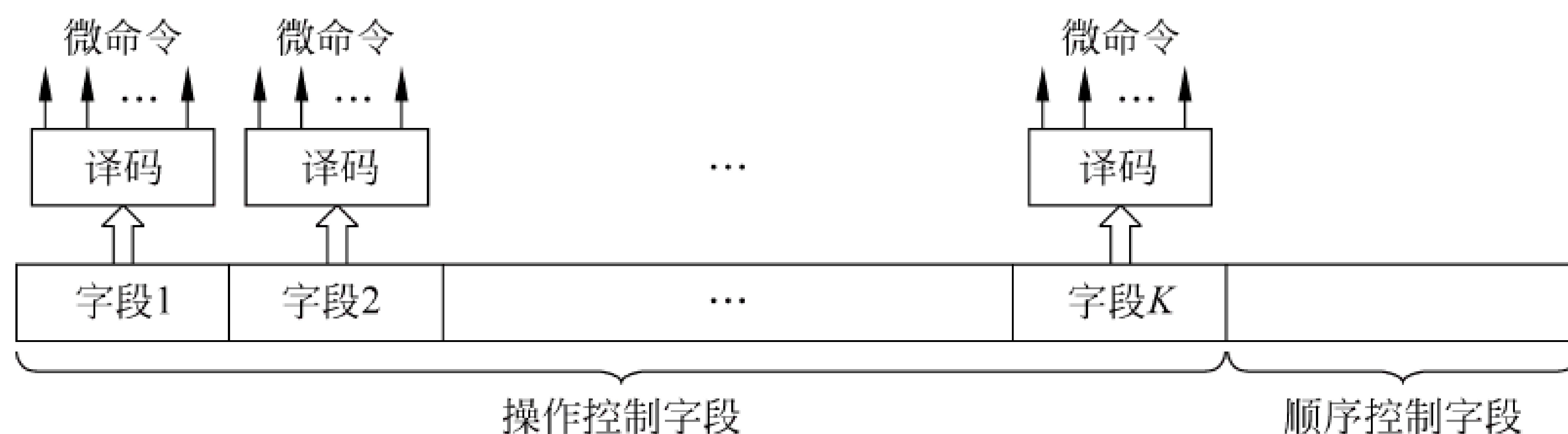


图 7-30　微指令结构——字段直接译码法

(1) 相斥性微命令分在同一字段内，相容性微命令分在不同字段内。前者可缩短微指令的字长，后者有利于实现并行操作，加快指令的执行速度。

(2) 一般将同类操作(或控制同一部件的操作)中互斥的微命令划分在一个字段内，如控制 ALU 操作的微命令+、−、Move 等划分在一个字段内。这样做使得微指令的结构清晰，易于编写微程序，也易于修改扩充功能。

(3) 每个字段包含的信息位不能太多，否则将增加译码线路的复杂性和译码时间。

字段译码方法既能缩短微指令字长度，又有并行操作效率高的优点，执行速度也比较快，因此，得到了广泛的应用，如 IBM 370 系列、VAX-11 系列机都采用此编码法。

3. 混合表示法

混合表示法是把直接表示法与字段直接译码法混合使用，即部分微命令用分段编码表示，将一些速度要求高，或与其他微命令都相容的用直接方式表示，其微指令结构如图 7-31 所示。混合表示法能综合考虑微指令字长、灵活性和执行微程序速度等方面的要求。

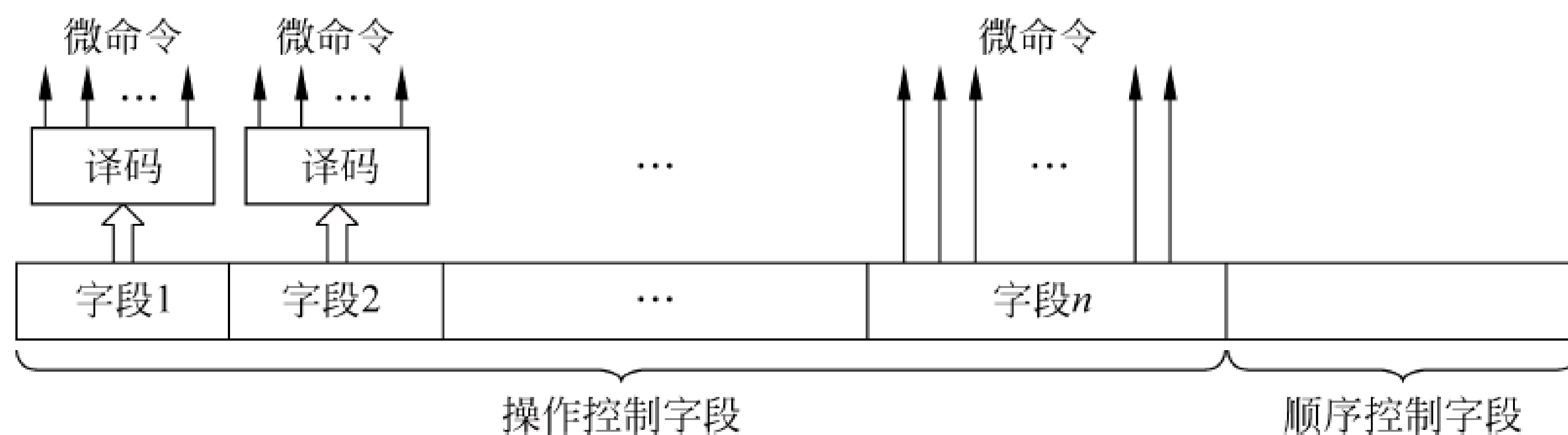


图 7-31　微指令结构——混合表示法

4. 常数字段控制法

另外，在微指令中还可以附设一个常数字段，就像指令中的立即数一样，来给某些执行部件直接发送常数。该常数字段可作为操作数送入 ALU 参与运算，也可作为计数器初值用来控制微程序循环次数。

【例 7.2】 表 7-5 中给出了 8 条微指令 $I_1 \sim I_8$ 所包含的微命令控制信号。试设计微指令操作控制字段的格式，要求所用的控制位最少，而且保持微指令本身内在的并行性。

解：

要使操作控制位最少，则应利用字段译码法对相斥性微命令进行编码；要保持并行性，则尽量对相容性微命令进行直接编码。在同一条微指令中出现的微命令即可认为它们可并行，因此可采用混合表示法进行编码。

表 7-5 8 条微指令 $I_1 \sim I_8$

微　指　令	所含微命令	微　指　令	所含微命令
I_1	ABCDE	I_5	CEGI
I_2	ADFG	I_6	AHJ
I_3	BH	I_7	CDH
I_4	C	I_8	ABH

观察表 7-5 可以发现 $I_1 \sim I_8$ 8 条微指令中共有 A、B、C、D、E、F、G、H、I、J 共 10 个微命令，注意观察表中各命令间的关系，尤其是两两互斥的。首先由 I_1 微指令可知，A、B、C、D、E 是相容微命令，任意两个都不能分在同一个字段。又通过比较，可知以下两组微命令两两互斥：{B、I、J}、{E、F、H}，所以，可按如图 7-32 所示的方式进行编码，只需 8 位，当然，结果不是唯一的，还可有其他的编码方式。

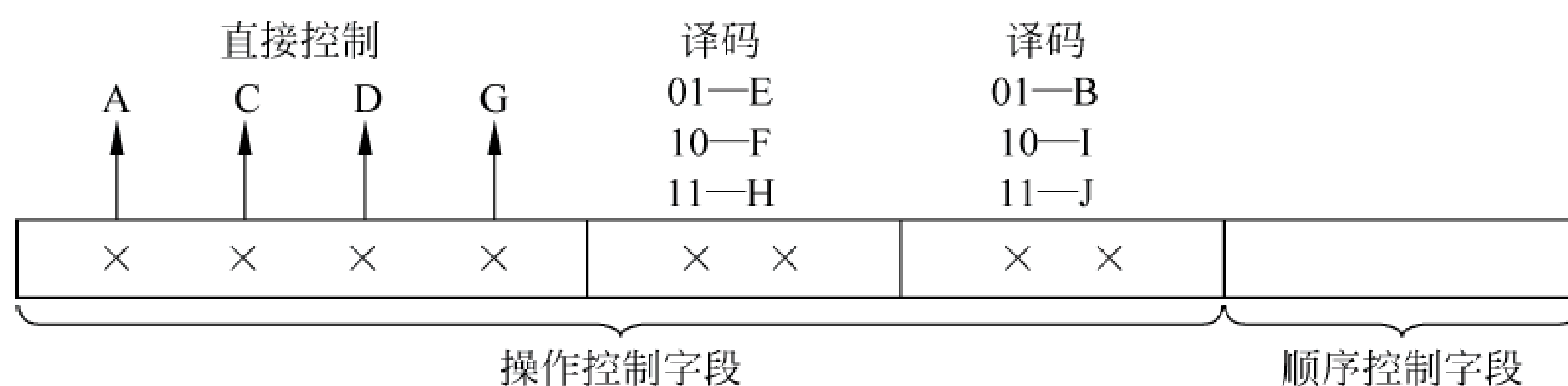


图 7-32 $I_1 \sim I_8$ 8 条微指令的结构

7.6.2 微地址的形成方法

微程序的执行过程如同程序的执行过程一样，也存在执行顺序的控制问题，所谓微程序的顺序控制是指在执行现行微指令的过程中或在执行完毕后，怎样控制产生下一条微指令的地址。下面讨论微指令中地址字段和后继微地址的形成问题。

1. 微程序入口地址的形成

由于每条机器指令都需要取指操作，所以将取指操作编制成一段公用微程序，通常安排在控存的 0 号或特定单元开始的一段控存空间内。

每一条机器指令对应着一段微程序，其入口就是初始微地址。首先由“取指令”微程序取出一条机器指令到 IR 中，然后根据机器指令操作码转换成该指令对应的微程序入口地址。这是一种多分支（或多路转移）的情况，常用以下三种方式形成微程序的入口地址。

1）一级功能转移

如果机器指令操作码字段的位数和位置固定，可以直接使操作码与入口地址码的部分位相对应。例如，某计算机有 16 条机器指令，指令操作码用 4 位二进制数表示，分别为 0000、0001、…、1111。现以字母 Q 表示操作码，令微程序的入口地址为 Q11B，例如 000011B 为 MOV 指令的入口地址，000111B 为 ADD 指令的入口地址，001011B 为 SUB 指令的入口地址……

由此可见，相邻两段微程序的入口地址相差 4 个单元，如图 7-33 所示。

2）二级功能转移

若各类指令的操作码的位数和位置不固定，需采用分级转移，第一次先按指令类型标志

转移，以区分出指令属于哪一类，如单操作数指令、双操作数指令等。在每一类机器指令中的操作码的位数和位置应当是固定的，第二次即可按操作码区分出具体是哪条指令，以便转移到相应微程序入口。

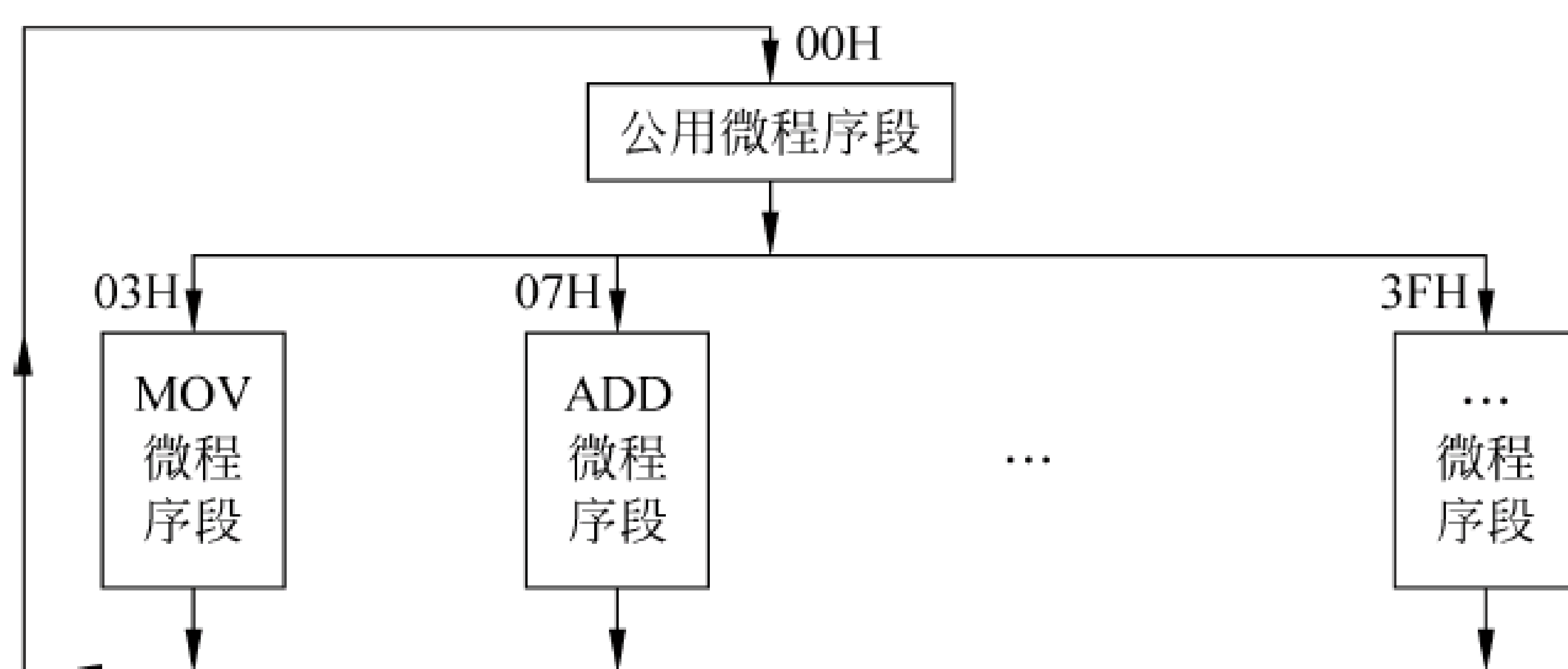


图 7-33　指令操作码与微程序入口地址

3）通过 PLA 电路实现功能转移

可编程逻辑阵列(PLA)实质上是一种译码-编码阵列，具有多个输入和多个输出，PLA 的输入是机器操作码和其他判别条件，PLA 的输出就是相应微程序的入口地址，这种方法对于变长度、变位置的操作码的处理更为有效而且转移速度较快。

2. 后继微地址的形成

在转移到一条机器指令对应的微程序入口地址后，则开始执行微程序，每条微指令执行完毕时，需根据其中的顺序控制字段的要求形成后继微指令地址。

1）增量方式(顺序-转移型微地址)

这种方式和机器指令的控制方式相类似，它也有顺序执行、转移和转子之分。顺序执行时，后继微地址就是现行微地址加上一个增量(通常为“1”)；转移或转子时，由微指令的顺序控制字段产生转移微地址。因此，微程序控制器中应当有一个微程序计数器(μPC)。为降低成本，一般情况下都是将微地址寄存器 μAR 改为具有计数功能的寄存器以代替 μPC。

在非顺序执行微指令时，用转移微指令实现转移。转移微指令的顺序控制字段分成两部分：转移控制字段(BCF)与转移地址字段(BAF)，如图 7-34 所示。由这两个字段结合，当转移条件满足时，将转移地址字段作为下一个微地址送 μPC；当转移条件不满足时，将直接从微程序计数器 μPC 中取得下一条微指令地址。

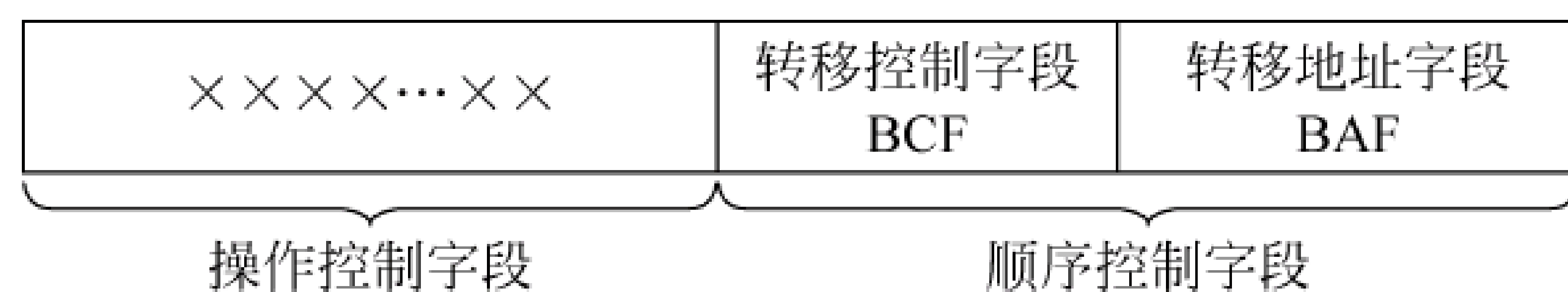


图 7-34　微指令结构——增量方式产生后继微地址

实现增量方式的微程序控制器如图 7-35 所示。其中，微程序计数器(μPC)的位数与控制存储器(CM)的容量相适应，由 μPC 给出 CM 地址，然后从 CM 中顺序读出微指令。

起始和转移地址发生器的功能有两个：一个功能是当一条新的机器指令装入 IR 时，它就形成机器指令的微程序段的起始地址且装入 μPC，而且随着节拍电位信号的到来，μPC

自动地增加一个增量，以便连续地从CM中读出微指令，相应的微操作控制信号按规定顺序发送到CPU的各个部分；另一个功能是当微指令指示其测试状态标志、条件代码或机器指令的某些位时，它就对指定的条件进行测试，若满足转移条件，就把新的转移地址装入μPC，实现转移；否则不装入新地址，微程序就顺序执行。所以，每次从CM中取出一条新的微指令时，μPC都增加，只有下列情况例外：

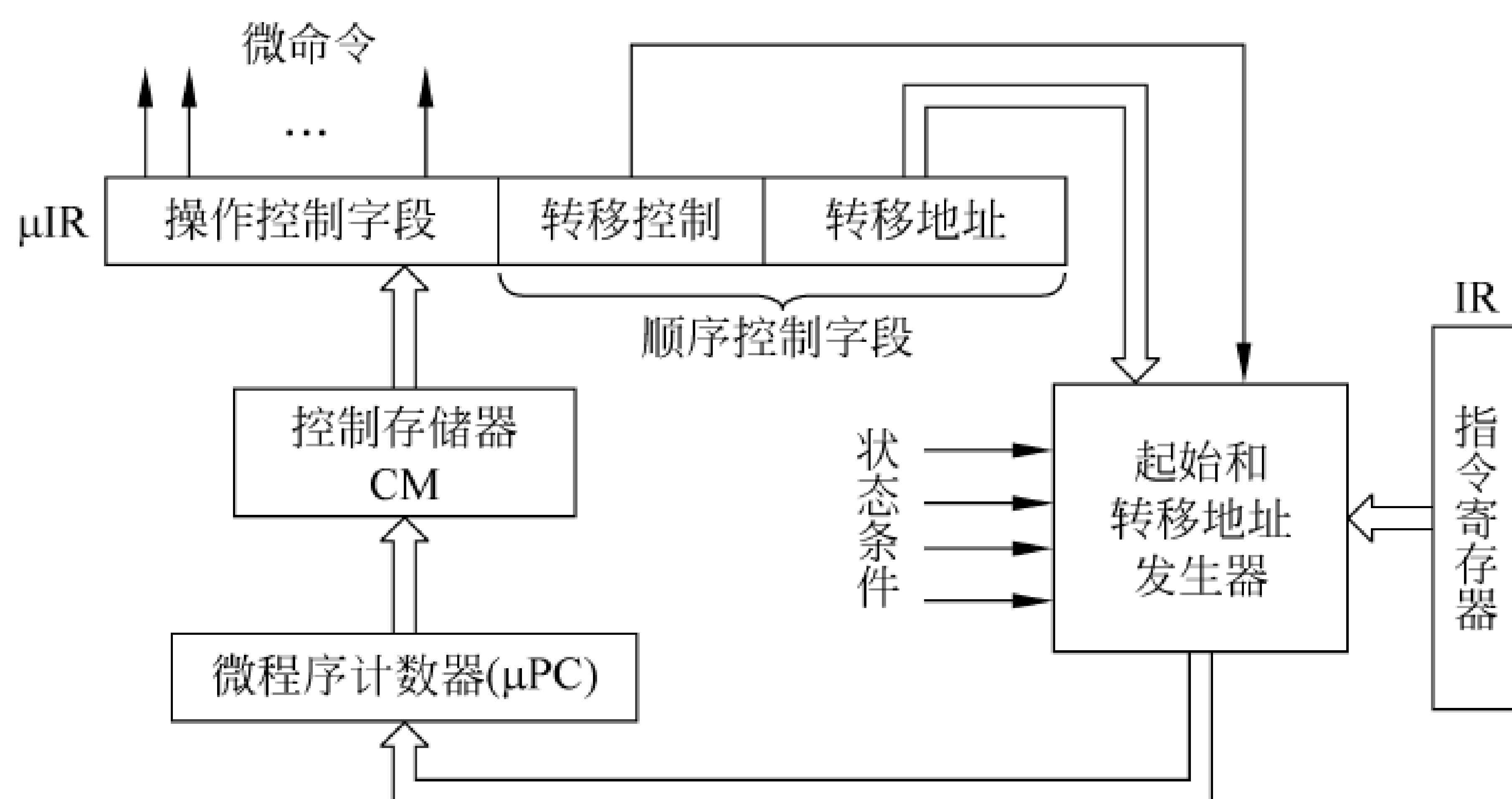


图 7-35 实现增量方式的微程序控制器

① 遇到END微指令时，就把“取指”微程序的入口地址装入μPC，开始取指令周期。

② 当一条新的指令装入IR时，就把该指令的微程序的入口地址装入μPC。

③ 遇到转移微指令且满足转移条件时，就把转移地址装入μPC。

增量方式的优点是简单，易于掌握，编制微程序容易，每条机器指令所对应的一段微程序一般安排在CM的连续单元中；其缺点是这种方式不能实现两路以上的并行微程序转移，因而不利于提高微程序的执行速度。

2）断定方式

断定方式与增量方式不同，它不采用μPC，微指令地址由微地址寄存器（μAR）提供。在微指令格式中，设置一个下地址字段，用于指明下一条要执行的微指令地址。当一条微指令被取出时，下一条微指令的地址（即下地址字段）送μAR。它相当于每条微指令都具有转移微指令的功能。采用这种方法就不必设置专门的转移微指令，但增加了微指令字的长度。

3）增量方式与断定方式的结合

这种控制方式中，微地址寄存器（μAR）有计数的功能（断定方式中的微地址寄存器（μAR）无计数功能），但在微指令中仍设置一个顺序控制字段，这是一种增量方式与断定方式相结合的方式。其顺序控制字段一般由两部分组成：顺序地址字段和测试字段。

① 顺序地址字段。可由设计者指定，一般是微地址的高位部分，用来指定后继微地址在CM中的某个区域内。

② 测试字段。根据有关状态的测试结果确定其地址值，一般对应于微地址的低位部分，相当于在指定区域内确定具体的分支。所依据的测试状态可能是指定的开关状态、指令操作码、状态字等。

测试字段如果只有一位，则微地址产生两路分支；若有两位，则最多可产生四路分支；以此类推，测试字段为n位为最多可产生2^n路分支。

③ 若无转移要求，则微地址寄存器计数得到后继微指令的地址。

【例 7.3】 微地址寄存器有 6 位($\mu A_5 \sim \mu A_0$),当需要修改其内容时,可通过某一位触发器的置 1 端 S 将其置"1"。现有以下三种情况:

(1) 执行"取指"微指令后,微程序按 IR 的 OP 字段($IR_3 \sim IR_0$)进行 16 路分支;

(2) 执行条件转移指令微程序时,按进位标志 C 的状态进行 2 路分支;

(3) 执行控制台指令微程序时,按 IR_4、IR_5 的状态进行 4 路分支。

请按多路转移方法设计微地址转移逻辑。

解:按所给设计条件,微指令的顺序控制字段有三种判别测试,分别为 P_1、P_2、P_3。由于修改 $\mu A_5 \sim \mu A_0$ 内容具有很大的灵活性,现分配如下:

(1) 用 P_1 和 $IR_3 \sim IR_0$ 修改 $\mu A_3 \sim \mu A_0$;

(2) 用 P_2 和 C 修改 μA_0;

(3) 用 P_3、IR_5、IR_4 修改 μA_5、μA_4。

另外还要考虑时间因素 T_4(假设 CPU 周期最后一个节拍脉冲),故转移逻辑表达式如下:

$$\mu A_5 = P_3 \cdot IR_5 \cdot T_4$$
$$\mu A_4 = P_3 \cdot IR_4 \cdot T_4$$
$$\mu A_3 = P_1 \cdot IR_3 \cdot T_4$$
$$\mu A_2 = P_1 \cdot IR_2 \cdot T_4$$
$$\mu A_1 = P_1 \cdot IR_1 \cdot T_4$$
$$\mu A_0 = P_1 \cdot IR_0 \cdot T_4 + P_2 \cdot C \cdot T_4$$

由于从触发器置 1 端 S 修改,故前 5 个表达式可用"与非"门实现,最后一个用"与或非"门实现。如图 7-36 所示即为微地址转移逻辑图。

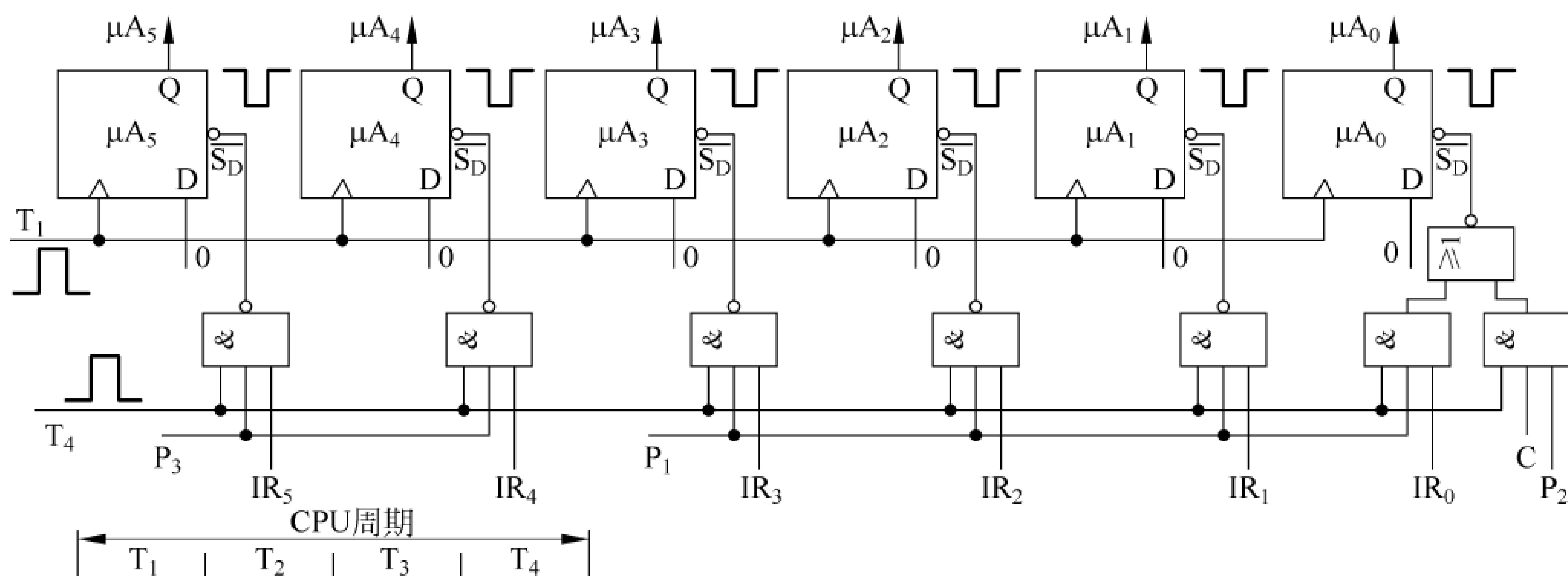


图 7-36 微地址转移逻辑图

7.6.3 微指令的格式及执行方式

1. 微指令的格式

微指令操作控制字段的表示方法是决定微指令格式的主要因素。在设计计算机时,考虑到速度、成本等原因,可采用不同的编码译码控制法,即在一台计算机中,也有几种编码译

码控制法同时存在的情况。微指令的格式大体分成两类：水平型微指令和垂直型微指令。

1）水平型微指令

一次能定义并执行多个并行操作微命令的微指令，叫做水平型微指令。水平型微指令的一般格式如图 7-37 所示。

控制字段	判别测试字段	后继地址字段

图 7-37 水平型微指令格式

按照控制字段的编码方法不同，水平型微指令又分为三种：第一种是全水平型（直接表示法）微指令，第二种是字段直接译码法水平型微指令，第三种是直接表示和字段译码相混合的水平型微指令。

2）垂直型微指令

微指令中设置微操作码字段，采用微操作码编译法来规定微指令功能的微指令，称为垂直型微指令。垂直型微指令的结构类似于机器指令的结构，它有微操作码，在一条微指令中只定义 1～2 个微操作命令，且每条微指令的功能简单。因此，实现一条机器指令的微程序要比水平型微指令编写的微程序长得多，它是采用较长的微程序结构去换取较短的微指令结构。

下面对 4 条垂直型微指令的微指令格式加以说明。设微指令字长为 16 位，微操作码 3 位（占高 3 位），可最多定义 8 种类型的微指令，只将其中典型的 4 种列出。

① 寄存器-寄存器传送型微指令。

寄存器-寄存器传送型微指令的格式如图 7-38 所示。

15　　13	12　　8	7　　3	2　　0
0 0 0	源寄存器编址	目标寄存器编址	其他

图 7-38 寄存器-寄存器传送型微指令的格式

其功能是把源寄存器数据送目标寄存器。其中 13～15 位为微操作码（下同）；源寄存器和目标寄存器编址各 5 位，可指定 31 个寄存器之一；第 0～第 2 位是其他字段，可协助本条微指令完成其他控制功能。

② 运算控制型微指令。

运算控制型微指令的格式如图 7-39 所示。

15　　13	12　　8	7　　3	2　　0
0 0 1	左输入源编址	右输入源编址	ALU

图 7-39 运算控制型微指令的格式

其功能是选择 ALU 的左、右两输入源信息，按 ALU 字段，即第 0～第 2 位所指定的 8 种运算操作中的一种功能进行处理，并将结果送入暂存器中。左、右输入源编址可指定 31 种信息源之一。

③ 访问主存微指令。

访问主存微指令的格式如图 7-40 所示。

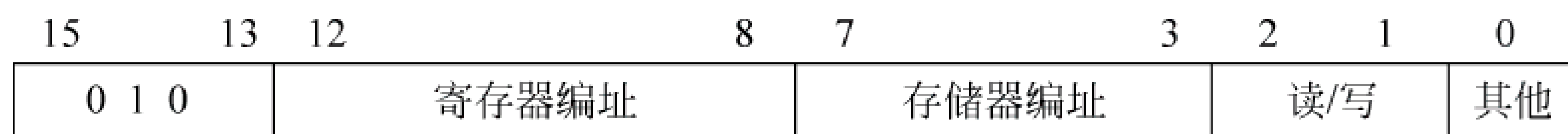


图 7-40　访问主存微指令的格式

其功能是将主存中一个单元的信息送入寄存器或者将寄存器的数据送往主存。其中，存储器编址是指按规定的寻址方式进行编址。第 1、第 2 位指定读操作或写操作，第 0 位可协助本条微指令完成其他控制功能。

④ 条件转移微指令。

条件转移微指令的格式如图 7-41 所示。

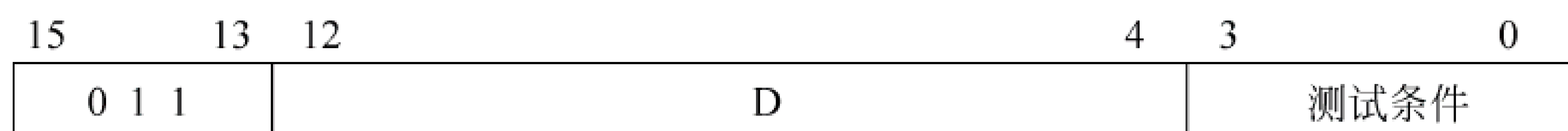


图 7-41　条件转移微指令的格式

其功能是根据测试对象的状态决定是转移到 D 所指定的微地址单元，还是顺序执行下一条微指令。9 位 D 字段不足以表示一个完整的微地址，但可以用来替代现行 μPC 的低位地址。测试条件字段有 4 位，可规定 16 种测试条件。除了这 4 种类型的微指令之外，还有移位控制型微指令、无条件转移微指令、其他微指令等。

2. 水平型微指令与垂直型微指令的比较

(1) 水平型微指令并行操作能力强，效率高，灵活性强；垂直型微指令则较差。

在一条水平型微指令中，设置有控制机器中信息传送通路以及进行所有操作的微命令，因此在进行微程序设计时，可以同时定义比较多地并行操作的微命令，控制尽可能多地并行信息传送，从而使水平型微指令具有效率高及灵活性强的优点。

在一条垂直型微指令中，一般只能完成一个微操作，控制一两个信息传送通路，因此垂直型微指令的并行操作能力弱，效率低。

(2) 水平型微指令执行一条机器指令的时间短，垂直型微指令执行的时间长。

因为水平型微指令的并行操作能力强，因此与垂直型微指令相比，可以用较少的微指令数来实现一条机器指令的功能，从而缩短了机器指令的执行时间。而且当执行一条微指令时，水平型微指令的微命令一般直接控制对象，而垂直型微指令要经过译码也会影响速度。

(3) 由水平型微指令解释指令的微程序，具有微指令字长比较长，而微程序较短的特点。垂直型微指令则相反，微指令字比较短而微程序较长。

(4) 水平型微指令用户难以掌握，而垂直型微指令与机器指令比较相似，相对来说较容易掌握。

水平型微指令与机器指令差别很大，一般需要对机器的结构、数据通路、时序系统以及微命令很精通才能进行设计。对机器已有的指令系统进行微程序设计是设计人员而不是用户的事情，因此这一特点对用户来讲并不重要，然而某些计算机允许用户自行设计并扩充指令系统，此时就要注意是否容易编写微程序的问题。

实际上，水平型和垂直型微指令两者之间并无明确的界限，一台机器的微指令也往往不局限于一种类型的微指令。混合型微指令兼有两者的特点，它可以采用不太长的微指令又

具有一定的并行控制能力,但微指令格式相对复杂。

3. 串行微程序控制方式和并行微程序控制方式

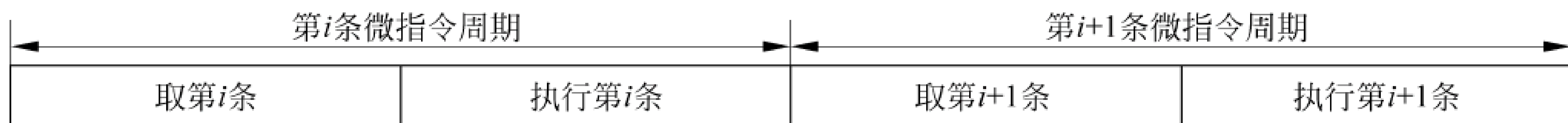
1) 串行微程序控制方式

在前面所讲微程序控制方式都属于串行微程序控制方式,所谓串行微程序控制方式,在执行当前微指令与取下一条微指令在时间上是顺序进行的,即只有当现行微指令执行完毕后,才能取下一条微指令,如图 7-42(a)所示。

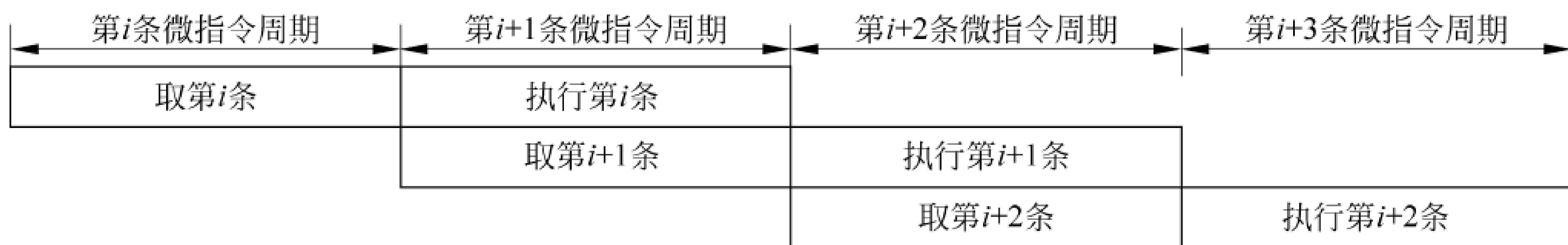
串行微程序控制中,微指令周期等于取指令的时间加上执行微指令时间之和,即等于只读存储器的读周期。串行微程序控制的微指令周期较长,但控制简单,形成微地址的硬件较少。

2) 并行微程序控制方式

因为在微指令周期中,取微指令和执行微指令分别由不同的计算机硬件来完成,不同的微指令的取指令和执行指令周期之间可以重叠(即并行工作),以缩短微指令周期。所谓并行微程序控制方式,就是将取微指令和执行微指令这两类操作在时间上重叠并行进行的方式,如图 7-42(b)所示。



(a) 串行微程序控制方式



(b) 并行微程序控制方式

图 7-42 串行微程序控制和并行微程序控制方式

在并行微程序控制方式中,要求在执行本条微指令的同时,预取下一条微指令,从而微指令周期仅等于执行微操作的时间,可以节约取微指令的时间。并行微程序控制可以缩短微指令周期,但是,为了不影响本条微指令的正确执行,需要增加一个微指令寄存器。

4. 动态微程序设计

微程序设计技术还有静态微程序设计和动态微程序设计之分。若一台计算机的机器指令只有一组微程序,而且这一组微程序设计好之后,一般无须改变而且也不好改变,这种微程序设计技术称为静态微程序设计。本节前面讲述的内容基本上属于静态微程序设计的概念。当采用 EPROM 作为控制存储器时,还可以通过改变微指令和微程序来改变机器的指令系统,这种微程序设计技术称为动态微程序设计。采用动态微程序设计时,微指令和微程序可以根据需要加以改变,因而可以在一台机器上实现不同类型的指令系统。这种技术又称为仿真其他机器指令系统,以便扩大机器的功能。

【例 7.4】 某微程序控制器中,采用水平型直接控制微指令格式断定方式。已知全机共有微命令 20 个,可判定的外部条件有 4 个,控制存储器容量为 256×32 位。

(1) 设计出微指令的具体格式；

(2) 画出该控制器的结构框图。

解：

(1) 微命令 20 个，采用水平型直接控制，所以操作控制字段需 20 位。可判定的外部条件有 4 个，所以判别测试字段为 4 位，控制存储器容量为 256 个单元，所以下地址字段需 8 位。由于控制存储器的字长为 32 位。操作控制字段用直接表示法，微指令格式如图 7-43 所示。

图 7-43 微指令的格式

(2) 对应上述微命令格式的微程序控制器逻辑框图，如图 7-44 所示。其中，微地址寄存器对应下地址字段，P 字段为判别测试字段。地址转移逻辑的输入是指令寄存器操作码，各状态条件以及判别测试字段所给的判别标志(某一位为 1)，其输入修改地址寄存器的适当位数，从而实现微程序的分支转移。

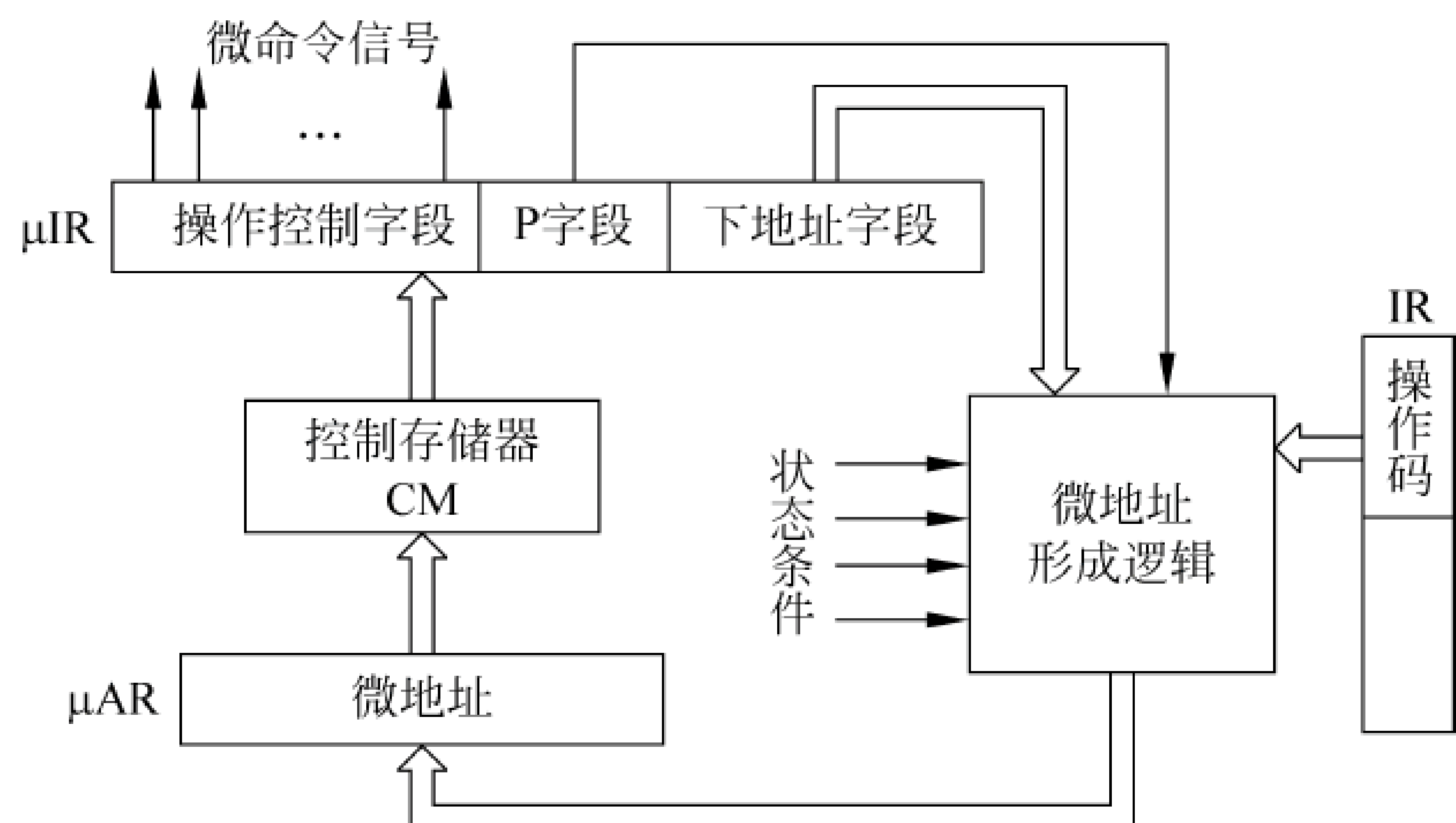


图 7-44 微程序控制器的逻辑框图

7.7 典型 CPU 及主要技术

前面的各个章节中，已经介绍了 CPU 的基本组成部分、工作原理及设计原理。下面将以 Intel 公司的 X86 CPU 为主线，以 Pentium 4 CPU 为例，简要介绍它们的结构、技术和原理。

7.7.1 Intel CPU

1. 80x86 CPU

1971 年，尚处于发展阶段的 Intel 公司推出了世界上第一个微处理器 4004。它是第一个用于计算器的 4 位微处理器。虽然微处理器中仅集成了 2300 个晶体管，速度很慢且功能有限，但是它是划时代的产品。

1978—1989 年，Intel 公司相继推出了 8086、80286、80386、80486 的 CPU，字长从 16 位

增加到32位，时钟频率从8086的4.77MHz提高到80486的100MHz，集成度突破了100万个晶体管的界限，80486集成了120万个晶体管。随着CPU技术的不断发展，Intel公司陆续研制出新型多款CPU，为了保证计算机能继续运行以往开发的各类应用程序，保护和继承丰富的软件资源，后续的CPU仍使用x86的基本指令集，从而形成了今天庞大的兼容CPU系列。

2. Pentium系列CPU

1993年，Intel公司继80486之后推出一种新的高速微处理器——Pentium，也就是人们所熟悉的“奔腾”处理器，它内含晶体管320万个，主频100/133/166MHz。

为了提高CPU的性能，Pentium CPU采用了下列技术。

1) 超标量流水线

超标量流水线指的是处理器中有多条流水线，并且在每个时钟周期中可同时获取一条以上的指令进行处理。

Pentium采用了超标量设计方法，它围绕两组通用寄存器设置了两条并行的整数流水线，即U指令流水线与V指令流水线。U流水线可执行所有整数与浮点数的指令，V流水线则可执行简单的整数指令与FXCH(交换寄存器内容)的浮点数指令，且每条流水线都有自己独立的算术逻辑单元、地址生成电路和数据高速缓冲存储器接口。这种设计使得在最佳的状况下，Pentium可以在一个时钟周期内同时并行地执行两条基本简单指令，比相同时钟频率的80486DX的性能提高了一倍。另外，在Pentium中还设置了一个浮点处理器(FPU)，其内部具有专用加、乘、除运算器。

2) 智能动态分支预测

智能动态分支预测是指在指令队列中，事先决定一个具体的分支指令是否应该被执行——即改变程序流向的一种处理器的“猜测”技术。

Pentium中提供了一个小的cache——分支目标缓冲器(Branch Target Buffer，BTB)来动态预测程序的分支。当一条指令导致程序分支时，BTB记忆这条指令和分支的目标地址，并根据这些信息来预测这条指令再次产生分支时的路径，然后直接从该处“预取”指令，保证流水线进行正常的“指令预取”。

3) 数据和指令分离的一级cache

采用双重分离式高速缓冲存储器设计。Pentium芯片内有两个8KB的高速缓冲存储器，一个用于存放指令，另一个用于存放数据，即双路高速缓冲结构，这种结构可以减少高速缓冲存储器发生争用的情况，并且两组存储器在需要时可随时提供服务，充分改善了微处理的性能。数据cache有两个接口，分别接U、V流水线，cache写操作和主存写操作同时进行，保证了数据的一致性。指令cache采用256位总线同指令预取缓冲器连接，一次可取32字节的指令操作码，从而大大加快了指令的预取速度。CPU可以同时访问两种高速缓冲存储器。

4) 总线吞吐能力的增强

Pentium的内部寄存器和数据总线仍为32位，外部数据总线则为64位，即微处理器与内存或I/O设备进行数据交换时的数据总线的宽度为64位，这样可使数据传输能力高达528MB/s，是80486的数据传输能力(105MB/s)的5倍，有效地平衡了外部数据传输慢而CPU内部数据传输快的矛盾，提高了CPU的性能。

5）处理器性能的增强

在 Pentium 的整数模块中，常用指令（如 MOV、DEC、PUSH、POP 等指令）改用硬件实现。同时对指令系统的微码做了重大的改进，使指令周期比 80486 大大缩短。

浮点单元采用独立的 8 级流水线 FPU，平均每个时钟周期完成一条浮点运算，浮点单元一些常用的指令（如 ADD、MUL、LOAD 等）用硬件实现。

继 Pentium 之后，Intel 公司又相继推出 Pentium Pro、Pentium MMX、Pentium Ⅱ、Pentium Ⅲ、Pentium 4 CPU。

Pentium Pro 是 Intel 公司于 1995 年 11 月推出的新一代高性能处理器——高能奔腾处理器，代号为 P6。采用 32 位寄存器、64 位外部数据总线和 36 位地址总线。并在 Pentium Pro 处理器基础上采用了超标量、14 级超级流水线设计，支持乱序执行，具有分支预测功能和推测执行功能。它内含三条流水线，将 80x86 的指令分解转换成 RISC 型微操作，即一个时钟周期内可处理三条 80x86 指令，且 Pentium Pro 片内集成了 256KB L2 高速缓存，使性能得到了进一步的提高。

Pentium MMX 是一种能直接进行多媒体信息处理的高效处理器，这款处理器并没有集成二级缓存，而是采用 MMX 技术来增强性能。MMX（MultiMedia eXtention）也可翻译为“多媒体扩展指令集”，是 Intel 公司为增强奔腾 CPU 在音像、图形和通信应用方面发明的一项多媒体增强指令集技术。MMX 处理器在原来 x86 指令集的基础上又扩展了 57 条指令，主要用于加速多媒体任务的执行。

1998 年，为抢占低端市场，Intel 推出了性价比较高的 Celeron（赛扬处理器）。有关 Celeron、Pentium Ⅱ、Pentium Ⅲ的特性，读者自己可参考相关资料。

2000 年 Intel 发布了 Pentium 4 处理器。Pentium 4 处理器集成了 4200 万个晶体管，到了改进版的 Pentium 4（Northwood）更是集成了 5500 万个晶体管；并且开始采用 0.18μm 进行制造，初始速度就达到了 1.5GHz。Pentium 4 还提供的 SSE2 指令集，这套指令集增加 144 个全新的指令。

2003 年 Intel 发布了 Pentium M（Mobile）处理器。Pentium M 处理器可提供高达 1.60GHz 的主频速度，并包含各种效能增强功能，如最佳化电源的 400MHz 系统总线、微处理作业的融合（Micro-OpsFusion）和专门的堆栈管理器（Dedicated Stack Manager），这些工具可以快速执行指令集并节省电力。

2005 年 Intel 推出的双核处理器有 Pentium D 和 Pentium Extreme Edition，同时推出 945/955/965/975 芯片组来支持新推出的双核处理器。Pentium D 处理器使用 Prescott 架构及 90nm 生产技术生产，其内核实际上由两个独立的 Prescott 核心组成，每个核心拥有独立的 1MB L2 缓存及执行单元。由于处理器中的两个核心都拥有独立的缓存，因此必须保证每个二级缓存当中的信息完全一致，否则就会出现运算错误。

单核心的 Pentium 4、Pentium 4 EE、Celeron D 以及双核的 Pentium D 和 Pentium EE 等 CPU 采用 LGA 775 封装。与以前的 Socket 478 接口 CPU 不同，LGA 775 接口 CPU 的底部没有传统的针脚，而代之以 775 个触点，即并非针脚式而是触点式，通过与对应的 LGA 775 插槽内的 775 根触针接触来传输信号。LGA 775 接口不仅能够有效提升处理器的信号强度、提升处理器频率，同时也可以提高处理器生产的良品率、降低生产成本。

3. Intel Core(酷睿)CPU

2005年至今是Intel Core(酷睿)系列微处理器时代。"酷睿"是一款节能的新型微架构,设计的出发点是提供出众的性能和能效。

2006年7月27日,Intel发布Core 2 Duo(酷睿2)。酷睿2是英特尔基于Core微架构的产品体系统称,它是一个跨平台的构架体系,包括服务器版、桌面版、移动版三大领域。其中,服务器版的开发代号为Woodcrest,桌面版的开发代号为Conroe,移动版的开发代号为Merom。

继LGA 775接口之后,Intel首先推出了LGA 1366平台,定位高端应用。首个采用LGA 1366接口的处理器代号为Bloomfield,采用经改良的Nehalem核心,基于45nm制程及原生四核心设计,内建8~12MB三级缓存。LGA 1366接口的处理器主要包括45nm Bloomfield核心酷睿i7四核处理器。随着Intel在2010年迈入32nm工艺制程,高端应用的代表被酷睿i7-980X处理器取代,全新的32nm工艺解决六核心技术,拥有最强大的性能表现。

Core i5是一款基于Nehalem架构的四核处理器,采用整合内存控制器,L3达到8MB,采用的是成熟的DMI(Direct Media Interface)总线,只支持双通道的DDR3内存,采用LGA 1156接口,32nm工艺制程,主流级别的代表有酷睿i5-650/760。

Core i3可看作Core i5的进一步精简版(或阉割版),有32nm工艺版本(研发代号为Clarkdale,基于Westmere架构)。Core i3最大的特点是整合GPU(图形处理器),由CPU+GPU两个核心封装而成,代表产品有酷睿i3-530/540。

2010年6月,Intel发布了第二代Core i3/i5/i7。第二代Core i3/i5/i7基于全新的Sandy Bridge微架构,它是取代Nehalem的一种新的微架构,采用32nm工艺制程、全新LGA 1155接口设计。相比第一代产品主要带来五点重要革新:采用全新32nm的Sandy Bridge微架构,更低功耗、更强性能。内置高性能GPU(核芯显卡),视频编码、图形性能更强。睿频加速技术2.0,更智能、更高效能。引入全新环形架构,带来更高带宽与更低延迟。全新的AVX、AES指令集,加强浮点运算与加密解密运算。

2012年4月Intel正式发布了Ivy Bridge(IVB)处理器。采用22nm工艺制程,3D晶体管技术,使CPU耗电量减少一半,继续使用LGA 1155接口。

2013年6月Intel发布了基于Haswell架构的新处理器,Haswell架构是用以取代Intel Ivy Bridge和Intel Sandy Bridge微架构的。Haswell和Ivy Bridge微架构一样,采用22nm制程,整体性能将比Ivy Bridge快两倍。

2014年8月,Intel正式发布了全新14nm芯片技术,并将其命名为Broadwell。在笔记本和平板市场,这些产品将被称做"酷睿M"。配备了Broadwell芯片(比如酷睿M)的平板电脑仅有3~5W的功耗,而纤薄的设计意味着可消除CPU的散热风扇。在图形性能方面,Broadwell的计算性能和采样相比Haswell分别提升了20%和50%,而视频质量引擎的提升更是达到了后者的两倍。

7.7.2 Pentium 4的结构

2000年Intel发布的Pentium 4处理器集成了4200万个晶体管,到了改进版的

Pentium 4(Northwood)集成了 5500 万个晶体管,开始采用 0.18μm 工艺制造,初始速度就达到了 1.5GHz。Pentium 4 也有对应型号的 Celeron 处理器,来应对低端市场。Socket 478 接口是 Pentium 4 系列处理器所采用的接口类型,针脚数为 478 针。基于 Socket 478 接口的 Pentium 4 处理器如图 7-45 所示。

基于 Pentium 4 处理器的个人电脑,可以让用户创建专业品质的影片,透过因特网传递电视品质的影像,并进行实时语音、影像通信,实时 3D 渲染,快速进行 MP3 编码解码运算,在连接因特网时运行多个多媒体软件。

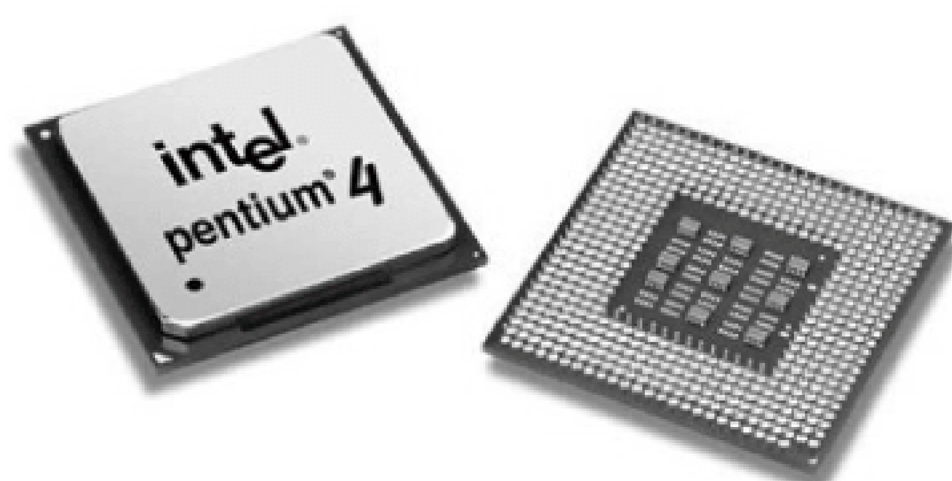


图 7-45　Pentium 4 处理器

Pentium 4 是一款 IA-32 体系结构的 32 位微处理器,也是第一款基于 Intel NetBurst (Intel Micro-architecture)微处理器,其逻辑功能图如图 7-46 所示。

总线接口单元　64位100MHz4倍速3.2GB/s
系统总线
L2缓存/缓存控制单元
256位 全速
指令TLB处
动态分支预测器4096个入口
指令寄存器
控存/微指令排序器
执行跟踪缓存 1200个μOPs
跟踪缓存分支预测：4096个入口的BTB
重排序缓冲区 ROB
资源分配/寄存器命名
整型/浮点μOP队列
存储μOP队列
整型运算调度
慢整型　快整型　快整型　Memory
浮点运算调度
FPGen　FPMem
整数运算寄存器文件
旁路网络
浮点运算寄存器文件
慢速 ALU
2XALU 简单指令
2XALU 简单指令
2XAGU 存入地址
2XAGU 读取地址
浮点存 浮点传送
浮点 乘另
SSE/ SSE2
MMX
256位 全速
L1数据缓存
8K字节，4路，双端口
硬件数据预取

图 7-46　Pentium 4 的逻辑功能

(1) 在 Pentium 4 中采用了一项非常重要的技术：流式 SIMD 扩展 2 技术(SSE2)。

SSE2 进一步对 MMX 技术和 SSE 扩展进行了增强。新增加了 144 条 SIMD 指令，包括浮点 SIMD 指令、整型 SIMD 指令、SIMD 浮点数和 SIMD 整型数相互转换的指令，以及在 XMM 和 MMX 寄存器之间的紧缩数据转换指令。SSE2 可同时使用以下类型的数据：4 个单精度浮点数、2 个双精度浮点数、16 个字节整数、8 个字整数、4 个双字整数、2 个四倍字整数、1 个 128 位的整数。

丰富的数据类型和新增加的 SSE2 指令集大大提高了 Pentium 4 对多媒体信息的处理能力。除了新增的 128 位 SIMD 指令外，Pentium 4 还兼容原来的 68 条 SIMD 指令. 并能使它们以更高的速度进行运算。且 SSE2 中还扩展了几条新的指令允许程序员控制数据的可缓存能力。

(2) Intel NetBurst 微体系结构。

① 快速执行引擎。Pentium 4 中有两组 2 倍速的 ALU 及两组 2 倍速的 AGU，一个用于存储地址的生成，一个则用于生成读取地址。另外还有一个慢速 ALU 用于执行复杂运算指令。4 个 ALU 和 AGU 均可在一个时钟周期内执行两次基本的整型运算，其执行速度可达 CPU 主频的 2 倍。

② 超流水线技术。Pentium 4 具有 20 级的流水线，它能使所有部件以极高的时钟频率全速运行。

③ 高级动态执行。乱序推测执行引擎保证同时执行的指令高达 126 条，流水线可同时执行 48 个读操作和 24 个写操作。且增强了分支预测的能力，采用的分支预测算法使准确度大大提高，分支目标缓存中的入口地址可达到 4096 个，并降低了流水线以预测失败时必须清除整个流水线造成的延迟。

④ 新型缓存体系结构。L1 缓存包括高级执行跟踪缓存和数据缓存。其中高级执行缓存用来存储已译码的指令，并把程序执行的控制流集中到一个单一的管线中，消除了主执行循环的译码延迟。而 8KB 的数据缓存采用了 4 路联合工作方式，并使用 64 字节的缓存管道，延迟时间低。

L2 缓存，Pentium 4 中加入了集成的 256KB 的 L2 缓存，也叫高级传输缓存，这种微体系结构从 Pentium Pro 开始就被引入 IA-32 处理器中，也称为 P6 处理器微体系结构，如图 7-47 所示。L2 缓存采用 8 路联合工作方式，按 2×64 字节进行组织，信息传输的宽度以 64 字节为单位，其时钟频率与 CPU 核心相同，与 CPU 内核连接的总线宽度为 256 位. 能实现与 CPU 的高速配合。

⑤ 高性能 4 倍速总线接口。

⑥ 超标量并行执行机制。

⑦ 使用一个扩充的硬件寄存器，通过重命名避免寄存器名字空间的限制。

⑧ 硬件指令预取。硬件指令预取部件能识别 CPU 内核执行程序的信息存取样本，并根据样本猜测下次要处理的信息，将其预先读入高速缓存中。

其中，BTB(Branch Target Buffer)是一个分支目标缓冲区，用来存放预测分支的所有可能目的地址。AGU(Addreess Generation Unit)即地址产生单元，它负责决定信息存取的正确地址。控存中存放的微指令的长度是固定的，很容易在执行流水线中处理。指令译码器则负责将机器指令转换成一条或多条微指令。

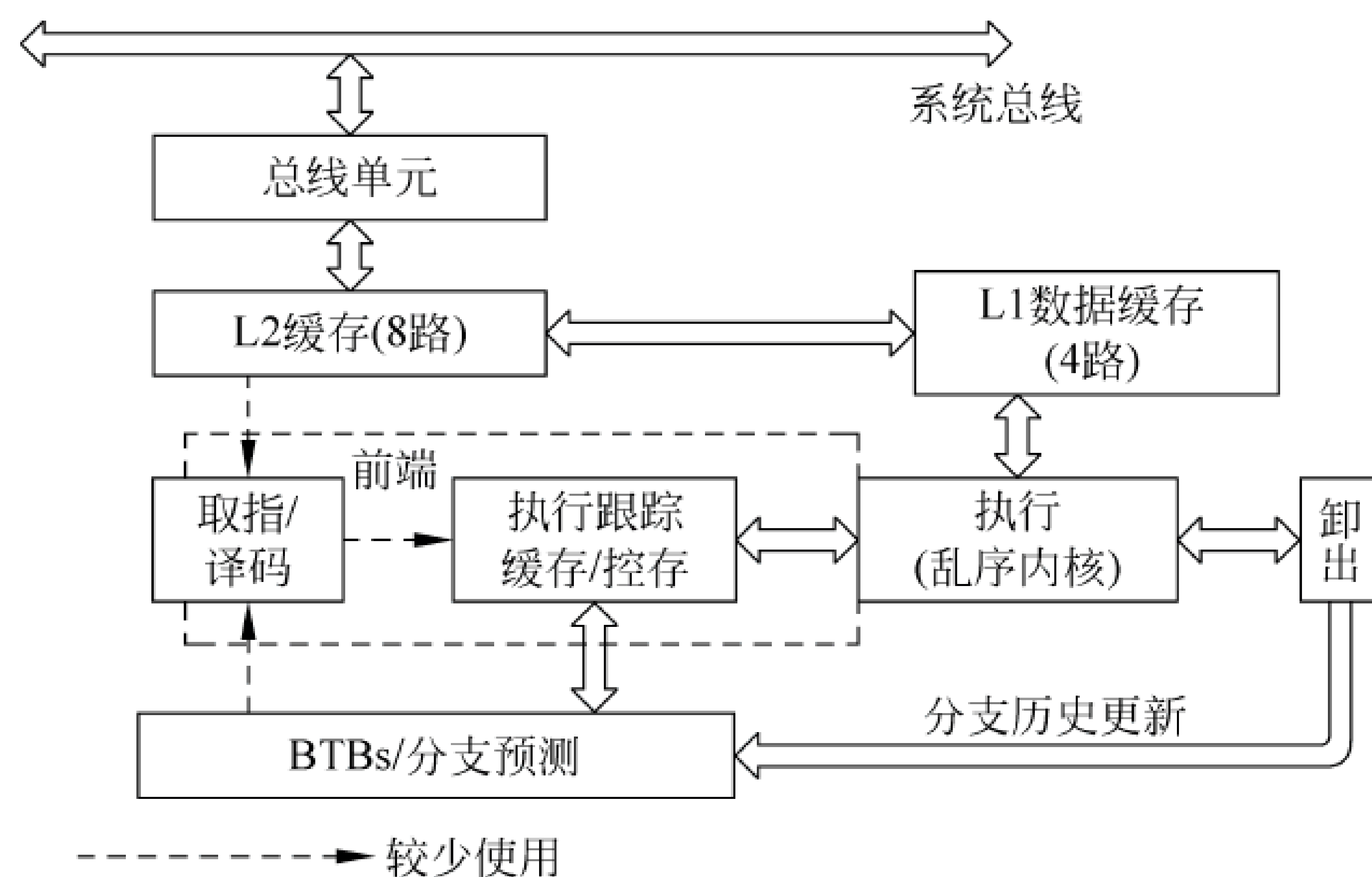


图 7-47 P6 处理器微体系结构

知识拓展

GPU 与 CPU

GPU(Graphic Processing Unit)即图像处理器,它是显卡的“心脏”,它决定了显卡的大部分性能。NVIDIA 公司在 1999 年发布 GeForce256 图形处理芯片时首先提出 GPU 的概念。GPU 能够从硬件上支持多边形转换与光源处理,计算多边形的 3D 位置和处理动态光线效果,也可以称为“几何处理”,实现三维图像和特效处理功能的“硬件加速”。

今天,GPU 已经不再局限于 3D 图形处理了,GPU 通用计算技术发展已经引起业界不少的关注。事实也证明,在浮点运算、并行计算等部分计算方面,GPU 可以提供数十倍乃至于上百倍于 CPU 的性能。GPU 通用计算方面的标准目前有 OPEN CL、CUDA、ATI STREAM。其中,开放计算语言(Open Computing Language,OpenCL)是第一个面向异构系统通用目的并行编程的开放式、统一的编程环境,便于软件开发人员为高性能计算服务器、桌面计算系统、手持设备编写高效轻便的代码,而且广泛适用于多核心处理器(CPU)、图形处理器(GPU)、Cell 类型架构以及数字信号处理器(DSP)等其他并行处理器,在游戏、娱乐、科研、医疗等各种领域都有广阔的发展前景。

虽然 GPU 能完成一些通用计算方面的任务,但是通用 CPU 所能完成的任务是 GPU 无法代替的,因此,不能用 GPU 来代替 CPU。Intel 和 AMD 意识到今后 CPU 和 GPU 只有走一条融合的道路才能在竞争中占得先机。

微软的 Windows 7 其中一个显著特性就是联合 GPU 和 CPU 的强大实力,组成了协同处理环境。CPU 运算非常复杂的序列代码,而 GPU 则运行大规模并行应用程序。微软利用 DirectX Compute 将 GPU 作为操作系统的核心组成部分之一,DirectX Compute 就是微软开发的 GPU 通用计算接口。

Intel 发布的酷睿二代 Sandy Bridge 处理器完整融合 GPU 和 CPU 在一块芯片上,显示核心将与 CPU 共用 L3 高速缓存;AMD Llano APU 已经将集成显卡提升到了一个新的高度。Intel Ivy Bridge 架构处理器的集成显示核心升格为 Intel HD Graphics 2500/4000,有 6~16 个 EU(运行单元),全面支持 DirectX 11、OpenGL 3.1 和 OpenCL 1.1。Intel Haswell 微架构处理器的自带 GPU 的性能可媲美同时代售价 50~70 美元的独立显示核心。

2014年8月,Intel发布了14nm工艺技术的芯片,并将其命名为Broadwell。相比Haswell,Broadwell在图形性能方面得到了大幅提升,视频质量是前者的两倍,Broadwell芯片将支持DirectX 11.2,以及4K和UHD分辨率。

本章小结

本章主要讲述了以下内容:

(1) CPU是计算机的中央处理部件,具有指令控制、操作控制、时间控制、数据加工等基本功能。

(2) 介绍了典型CPU的组成、工作原理及特点,早期的CPU由运算器和控制器两大部分组成。随着高密度集成电路技术的发展,当今的CPU芯片变成运算器、cache和控制器三大部分,其中还包括浮点运算器、存储管理部件等。CPU至少要有以下6类寄存器:指令寄存器、程序计数器、地址寄存器、缓冲寄存器、通用寄存器、状态条件寄存器。

(3) CPU从存储器取出一条指令并执行这条指令的时间和称为指令周期。由于各种指令的操作功能不同,各种指令的指令周期是不尽相同的。划分指令周期,是设计操作控制器的重要依据。

(4) 时序信号产生器提供CPU周期(也称机器周期)所需的时序信号。操作控制器利用这些时序信号进行定时,有条不紊地取出一条指令并执行这条指令。

(5) 微程序设计技术是利用软件方法设计操作控制器的一门技术,具有规整性、灵活性、可维护性等一系列优点,因而在计算机设计中得到了广泛的应用,并取代了早期采用的硬布线控制器设计技术。但是随着VLSI技术的发展和对机器速度的要求,硬布线逻辑设计思想又得到了重视。硬布线控制器的基本思想是:某一微操作控制信号是指令操作码译码输出、时序信号和状态条件信号的逻辑函数,即用布尔代数写出逻辑表达式,然后用门电路和触发器等器件实现。

习题七

一、名词解释

1. 指令周期　2. 机器周期　3. 时钟周期　4. 节拍
5. 微操作　6. 微命令　7. 微程序　8. 微指令

二、选择题

1. 硬布线控制器也称为(　　)。
   A. 存储逻辑控制器　B. 运算器
   C. 微程序控制器　D. 组合逻辑控制器

2. 相对于微程序控制器,硬布线控制器的特点是(　　)。(2009年全国硕士研究生入学统考计算机学科专业试题)
   A. 指令执行速度慢,指令功能的修改和扩展容易
   B. 指令执行速度慢,指令功能的修改和扩展难

C. 指令执行速度快，指令功能的修改和扩展容易

D. 指令执行速度快，指令功能的修改和扩展难

3. 在 CPU 中，用来指定下一条要执行指令的地址的寄存器是（　　）。

A. 地址寄存器　　B. 指令寄存器

C. 程序计数器　　D. 状态条件寄存器

4. 状态条件寄存器的主要用途是用来存放（　　）。

A. 运算类型

B. 逻辑运算结果标志

C. 算术运算结果标志

D. 算术、逻辑运算及测试指令结果标志和 CPU 当前的状态标志

5. 一般来说，和微指令的执行周期相对应的是（　　）。

A. 时钟周期　　B. 机器周期　　C. 指令周期　　D. 节拍周期

6. 微程序存放在（　　）中。

A. 主存储器　　B. 控制存储器　　C. 指令寄存器　　D. 通用寄存器

7. 将微程序存储在 E^2PROM 中并且可以进行修改称为（　　）。

A. 静态微程序设计　　B. 动态微程序设计

C. 汇编程序设计　　D. 毫微程序设计

8. 在微程序控制器中，机器指令和微指令的关系为（　　）

A. 一条微指令由若干条机器指令组成

B. 每条机器指令由一条微指令解释执行

C. 每一段微程序由一条机器指令解释执行

D. 每条机器指令由一段微指令解释执行

9. 关于微指令的编码方式，下列说法正确的是（　　）。

A. 字段编译码表示法的微指令位数多

B. 直接表示法的微指令位数多

C. 直接表示法和字段编译码表示法不影响微指令的长度

D. 以上说法都不对

10. 水平型微指令和垂直型微指令的差别在于（　　）。

A. 一条水平型微指令只能完成一个微操作

B. 一条垂直型微指令完成多个并行微操作

C. 两者都能一次完成多个微操作

D. 垂直型微指令编码长度一般比水平型微指令短

11. 某计算机采用微程序控制器，共有 32 条指令，公共的取指令微程序包含 2 条微指令，各指令对应的微程序平均由 4 条微指令组成，采用断定法（下址字段法）确定下条微指令的地址，则微指令中下址字段的位数至少是（　　）位。（2014 年全国硕士研究生入学统考计算机学科专业试题）

A. 5　　B. 6　　C. 8　　D. 9

12. 某计算机的控制器采用微程序控制器方式，微指令中的操作控制字段采用编译码法，共有 33 个微命令，构成 5 个互斥类，分别包含 7、3、12、5 和 6 个微命令，操作控制字段的

位数至少有(　　)位。(2012 年全国硕士研究生入学统考计算机学科专业试题)

A. 5　　B. 6　　C. 15　　D. 33

13. 下列关于 RISC 的叙述中,错误的是(　　)。(2009 年全国硕士研究生入学统考计算机学科专业试题)

A. RISC 一定采用微程序控制器

B. RISC 大多数指令的执行在一个 CPU 周期内完成

C. RISC 的内部通用寄存器数量相对 CISC 多

D. RISC 的指令数、寻址方式和指令格式种类相对 CISC 少

三、综合题

1. CPU 的基本功能是什么,一个典型的 CPU 至少由哪些部件组成?

2. 试对微程序控制器和硬布线控制器进行比较。

3. 计算机为什么要设置时序系统,说明指令周期、机器周期及时钟周期的含义。

4. CPU 的控制方式有哪几种?试说明它们的特点。

5. 某 CPU 的时钟频率为 100MHz,其时钟周期是多少?若已知每个机器周期平均包含 2 个时钟周期,该机的平均指令执行速度为 25MIPS,试问:

(1) 平均指令周期是多少 ns?

(2) 平均每个指令周期含有多少个机器周期?

(3) 若要得到 50MIPS 的指令执行速度,则应采用主频率为多少 MHz 的 CPU 芯片?

6. CPU 结构如图 7-48 所示,其中有一个累加寄存器(AC),一个状态条件寄存器,各部分之间的连线表示数据通路,箭头表示信息传送方向。

(1) 标明图 7-48 中 4 个寄存器的名称;

(2) 简述指令从主存取到控制器的数据通路;

(3) 简述数据在运算器和主存之间进行存/取访问的数据通路。

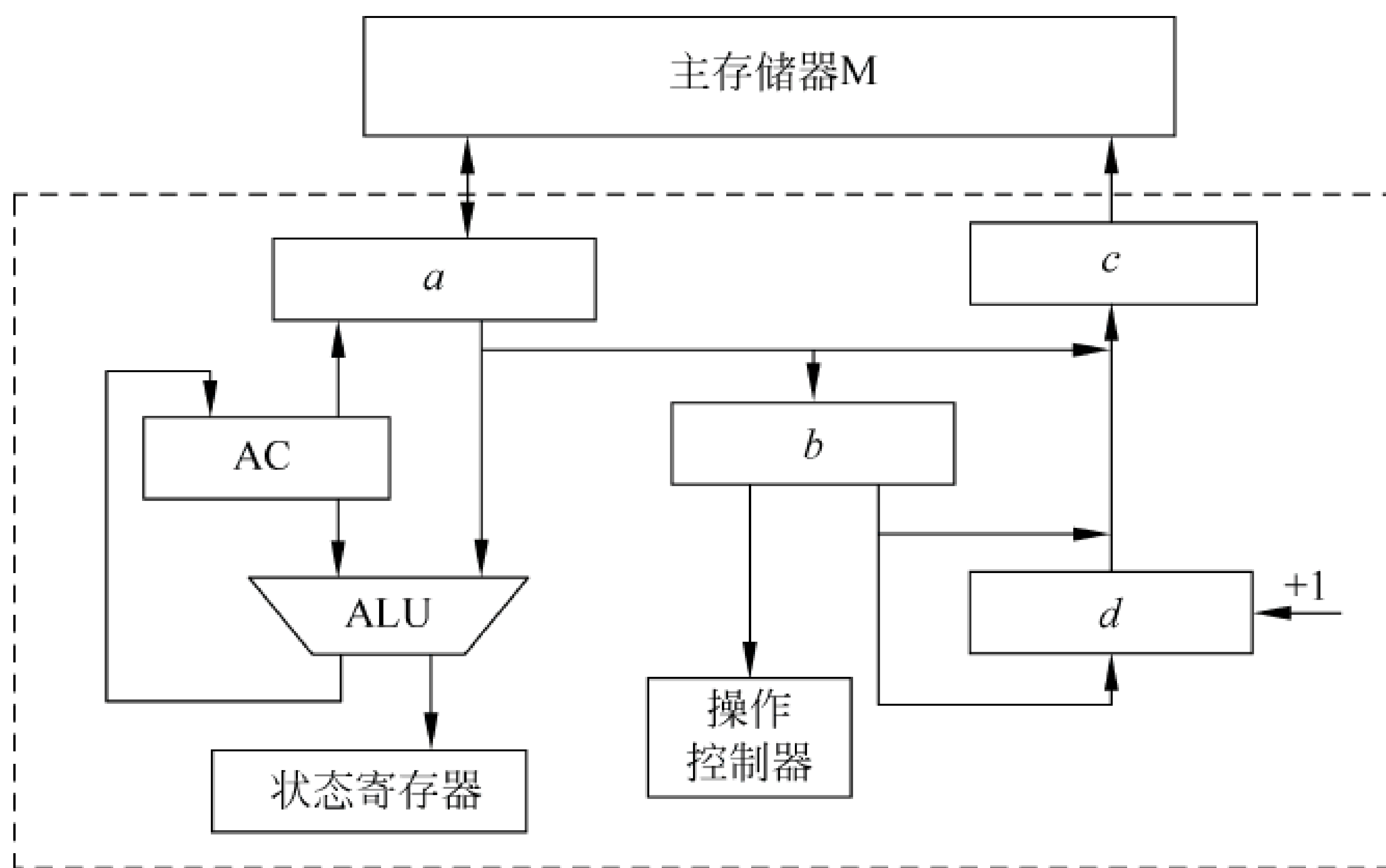


图 7-48　CPU 的结构图

7. 某计算机有以下部件:ALU、移位器、主存 M、主存数据寄存器 MDR、主存地址寄存器 MAR、指令寄存器 IR、通用寄存器 R_0~R_3、暂存器 C 和 D(见图 7-49)。

(1) 请将各逻辑部件组成一个数据通路,并标明数据流向;

(2) 画出 ADD R_1,(R_2)指令的指令周期流程图,指令功能是(R_1)+((R_2))→R_1。

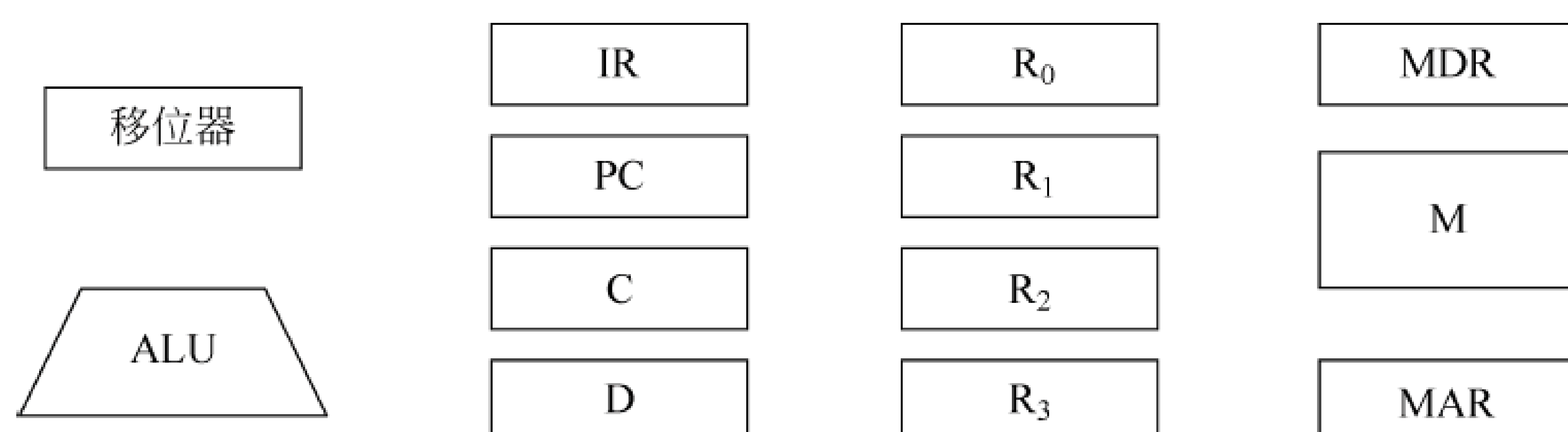


图 7-49　逻辑部件

8. 图 7-50 为双总线结构 CPU 的数据通路,IR 为指令寄存器,PC 为程序计数器(具自增功能),M 为主存(受 R/W 控制),AR 为地址寄存器,DR 为数据缓冲寄存器,ALU 由加、减控制信号决定完成何种操作,G 控制的是一个门电路,另外的输入、输出等控制信号已由图中标明。

(1) SUB R_1,R_3 指令完成 R_3←(R_3)−(R_1)的操作;

(2) LDA (R_2),R_0 指令完成将(R_2)为地址的主存单元的内容取至寄存器 R_0 中,画出其指令周期流程图,并列出相应的微操作控制信号(假定指令地址已送入 PC)。

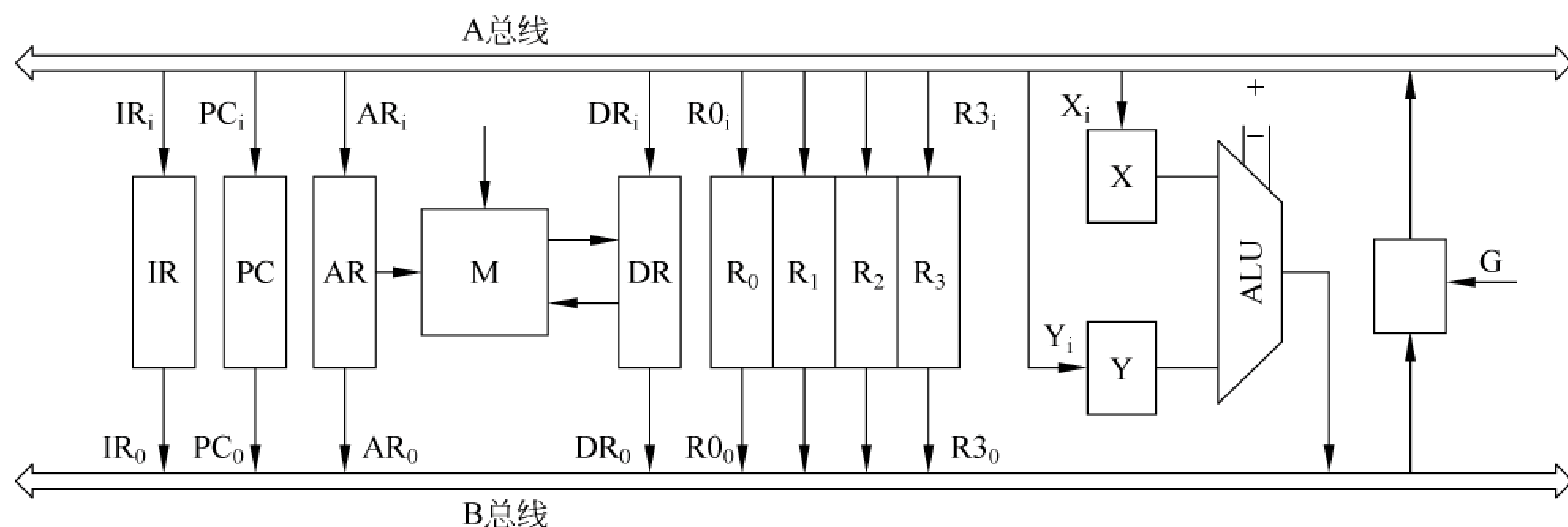


图 7-50　双总线结构 CPU 的数据通路

9. 运算器结构如图 7-51 所示,R_1,R_2,R_3 是三个寄存器,A 和 B 是两个三选一的多路开关,通路的选择由 AS_0、AS_1 和 BS_0、BS_1 端控制,例如 BS_0BS_1=11 时,选择 R_3,BS_0BS_1=01 时,选择 R_1……ALU 是算术/逻辑单元,S_1S_2 为它的两个操作控制端,其功能如下:

$$S_1S_2 = 00 \text{ 时,ALU 输出} = A$$

$$S_1S_2 = 01 \text{ 时,ALU 输出} = A + B$$

$$S_1S_2 = 10 \text{ 时,ALU 输出} = A - B$$

$$S_1S_2 = 11 \text{ 时,ALU 输出} = A \oplus B$$

请设计控制运算器通路的微指令格式。

10. 已知某机采用微程序控制方式,其控制存储器容量为 512×48(位),微程序在整个控制存储器中实现转移,可控制的条件共 4 个,微指令采用水平型格式,后继微指令地址采用断定方式,如图 7-52 所示。

(1) 微指令中的三个字段分别对应多少位?

(2) 画出对应这种微指令格式的微程序控制器逻辑框图。

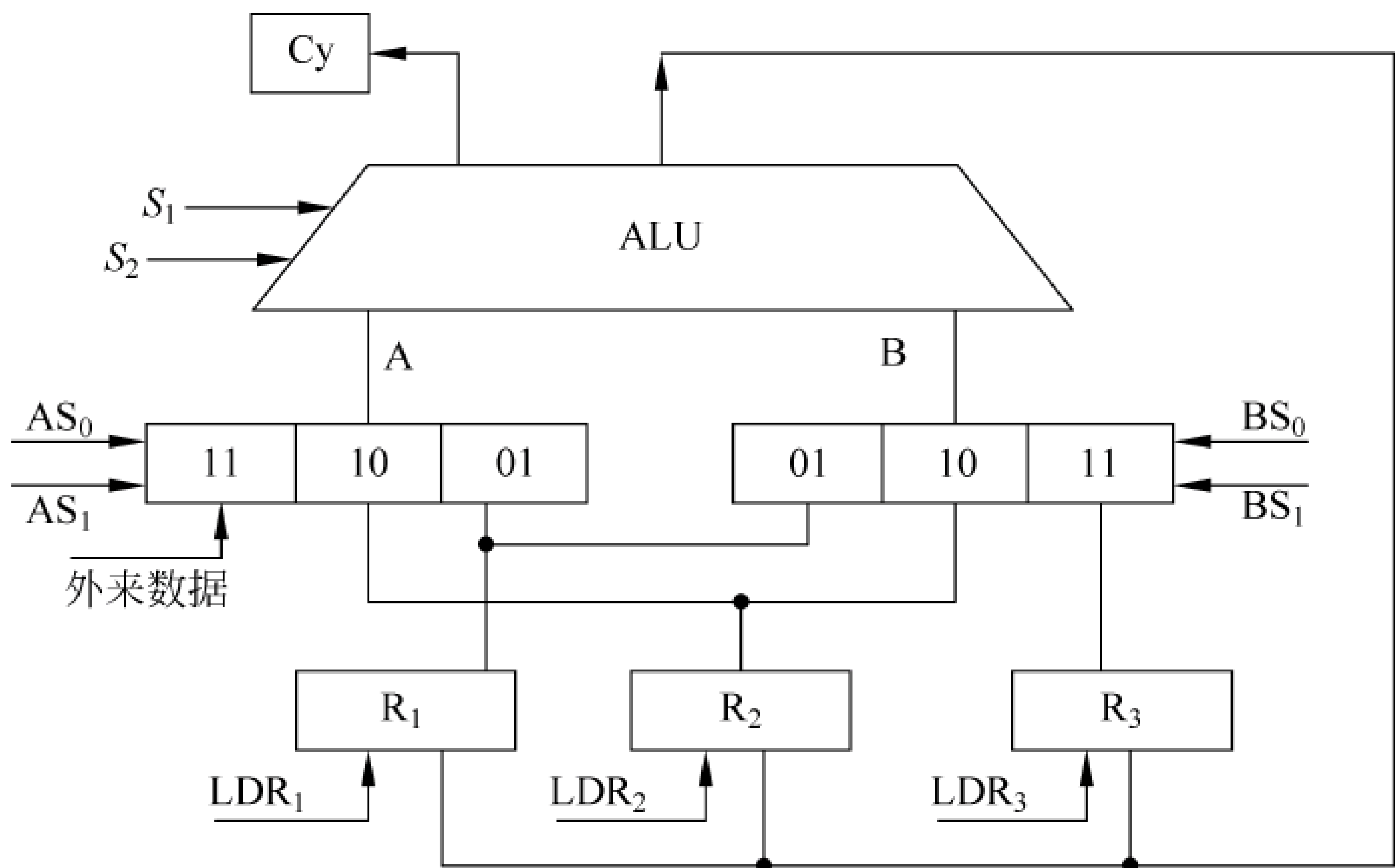


图 7-51 运算器结构

图 7-52 微指令格式

11. 假设某计算机的运算器框图如图 7-53 所示，其中 ALU 为 16 位的加法器(高电平工作)，SA、SB 为 16 位锁存器，4 个通用寄存器由 D 触发器组成，Q 端输出，其读写控制如下表所示。

读控制

R	RA_0	RA_1	选择
1	0	0	R_0
1	0	1	R_1
1	1	0	R_2
1	1	1	R_3
0	×	×	不读出

写控制

W	WA_0	WA_1	选择
1	0	0	R_0
1	0	1	R_1
1	1	0	R_2
1	1	1	R_3
0	×	×	不写入

要求：(1) 设计微指令格式；

(2) 画出 ADD，SUB 两条微指令程序流程图(不编码)。

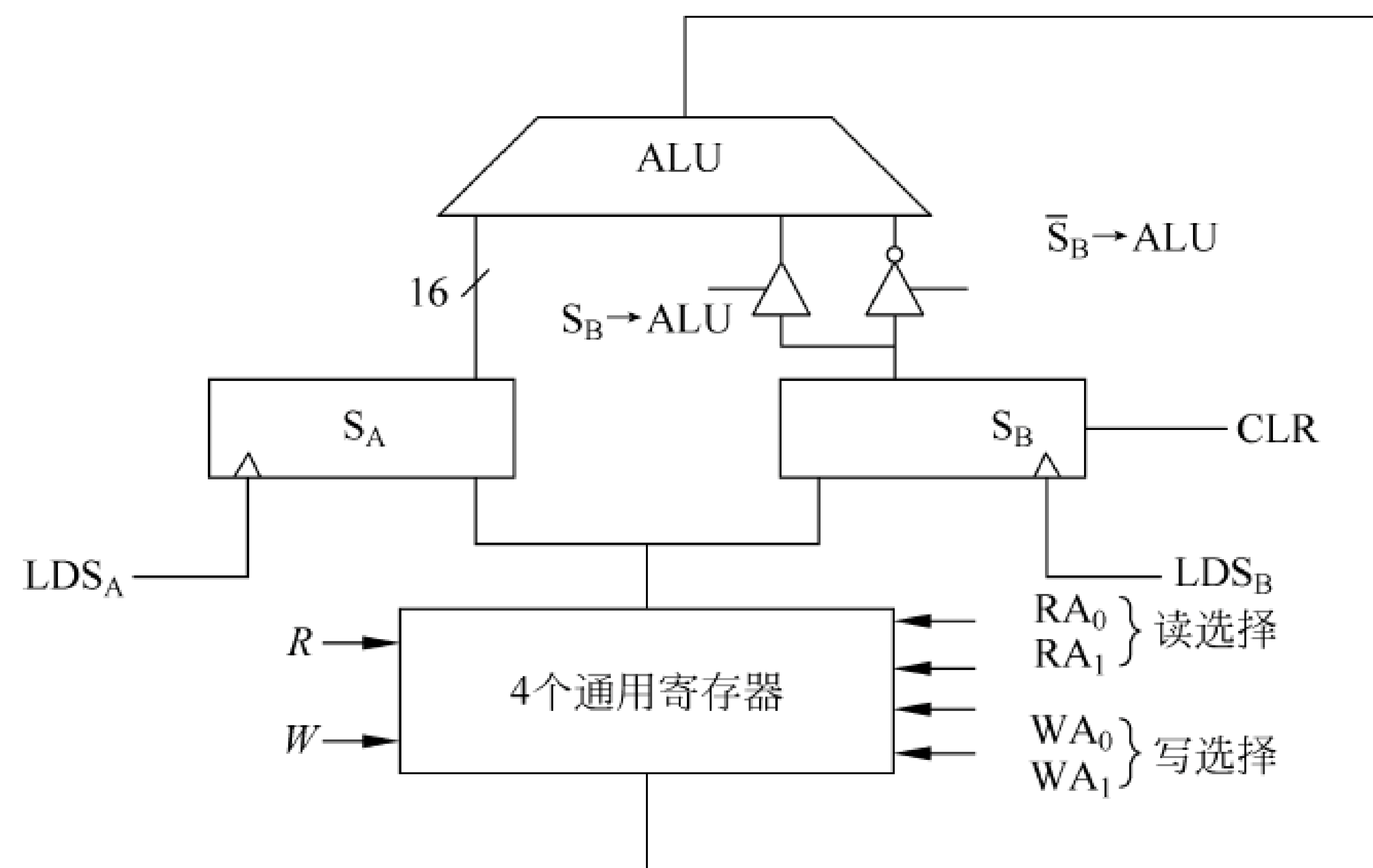


图 7-53　运算器框图

12. 已知 MOV、ADD、COM、ADT 4 条指令微程序流如图 7-54 所示，已知 P(1)的条件是指令寄存器 OP 字段，P(2)的条件码是进位寄存器 CJ，请设计画出微程序控制器地址转移逻辑图。

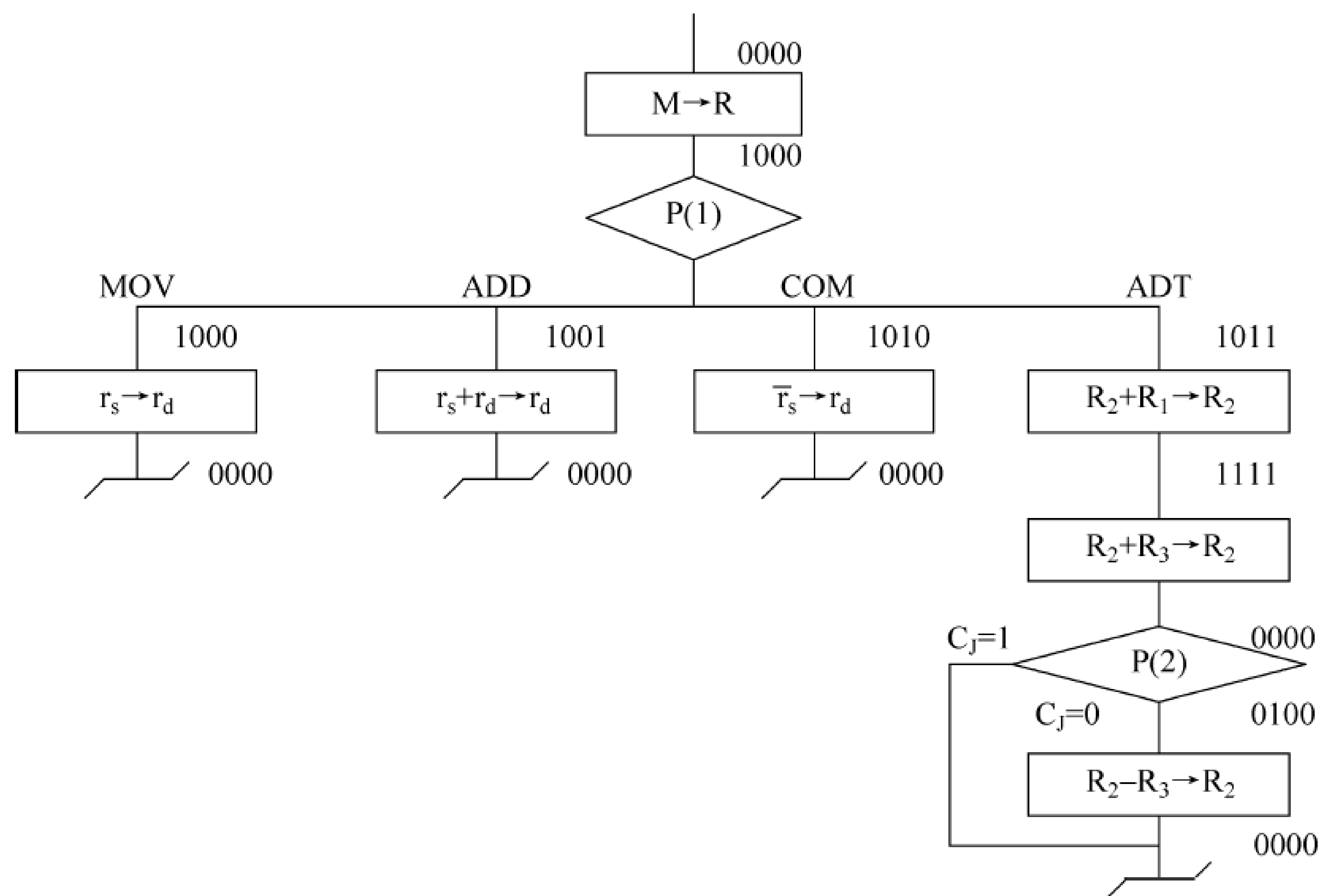


图 7-54　4 条指令框图

第8章 并行组织与结构

由于器件本身的限制，任何单处理机的速度最高不能超过某个上限值。要突破这个界限，采用多处理机或多计算机的并行体系结构是行之有效的途径。现代计算机的设计者，无论是在 CPU 内部还是在计算机体系结构上，越来越多地采用并行处理技术和并行体系结构。

本章首先介绍计算机系统的并行性概念，对计算机中使用的时间重叠、资源重复和资源共享等提高并行性的技术途径进行概括性的介绍；然后分别介绍现代计算机普遍采用的流水线技术和多处理机技术等并行处理技术；最后对近几年发展起来且应用非常广泛的机群系统、多核处理器进行讨论。

8.1 计算机系统的并行性

现代计算机系统是由硬件和软件组成的复杂系统。当设计一种新型计算机系统时，首先面临的问题是什么呢？接下来会列出很多问题，如机器指令系统的设计、功能组织、逻辑设计、实现技术等。实现技术又包括集成电路设计、制造和封装技术、系统制造和组装技术、供电、冷却技术等，而最重要的是确定计算机的体系结构。本节首先介绍计算机系统结构的概念及分类，然后介绍提高机器性能所采用的时间并行和空间并行的技术途径。

8.1.1 计算机体系结构的概念

“计算机体系结构”这个名词来源于英文 Computer Architecture，也有译成“计算机系统结构”的。Architecture 这个词原来用于建筑领域，其意义是“建筑学”、“建筑物的设计或式样”，它是指一个系统的外貌。20 世纪 60 年代这个词被引入计算机领域，“计算机体系结构”一词已经得到普遍应用，它研究的内容不但涉及计算机硬件，也涉及计算机软件，已成为一门学科。但对“计算机体系结构”一词的含义仍有多种说法，并无统一的定义。

经典的“计算机体系结构”定义是 C. M. Amdahl 等人在 1964 年设计 IBM 360 系统时提出的。他们把计算机体系结构定义为程序员所看到的一个计算机系统的概念性结构和功能特性。这实际上是计算机系统的外特性。

按照计算机层次结构，不同程序设计者所看到的计算机有不同的属性。例如，对于使用高级语言的程序员来讲，一台 IBM 3090 大型机、一台 VAX-11/780 小型机或一台 PC 微型机，看起来都是一样的，因为在这三台计算机上运行他所编制的程序，所得到的结果是一样

的。但对于使用汇编语言程序的程序员来讲，由于这三台机器的汇编语言指令完全不一样，他所面对的计算机的属性就也会不一样。另外，即使对同一台机器来讲，处在不同级别的程序员，例如应用程序员、高级语言程序员、系统程序员和汇编程序员，他们所看到的计算机外特性也是完全不一样的。那么通常所讲的计算机体系结构的外特性应是处在哪一级的程序员所看到的外特性呢？比较一致的看法是机器语言程序员或编译程序编写者所看到的外特性，这种外特性是指由他们所看到的计算机基本属性，即计算机的概念性结构和功能特性，这是机器语言程序员或编译程序编写者为使其所编写、设计或生成的程序能在机器上正确运行所必须遵循的。由机器语言程序员或编译程序编写者所看到的计算机的基本属性是指传统机器级的体系结构，在传统机器级之上的功能被视为属于软件功能，而在其之下的则属于硬件和固件功能。因此，计算机系统的概念性结构和功能属性实际上是计算机系统中软硬件之间的界面。

在计算机技术中，一种本来存在的事物或属性，但从某种角度看似乎不存在，称为透明性现象。通常，在一个计算机系统中，低层机器级的概念性结构和功能特性，对高级语言程序员来说是透明的。由此看出，在层次结构的各个级上都有它的体系结构。

就目前的通用机来说，计算机体系结构的属性应包括以下几个方面：

(1) 机器的数据表示，即机器硬件能直接识别和处理的数据类型。

(2) 寄存器组，包括各种寄存器的定义、数量和使用方式等。

(3) 指令系统，包括机器提供的各种指令集及寻址方式等。

(4) 数据通路，机器不同部件进行数据传输的通路。

(5) 中断系统，包括中断的类型和中断的处理方法等。

(6) 机器状态，如管态、目态及各种状态之间的切换等。

(7) 存储系统，主存容量，程序员可用的最大存储空间和存储保护等。

(8) 输入输出，输入输出的连接方式、输入输出的传输控制方式等。

现代计算机体系结构的概念，除了包括经典的计算机体系结构的概念范畴(指令集结构)，还包括计算机组成和计算机实现的内容。

计算机组成(Computer Organization)所研究的是计算机系统的逻辑实现，而计算机实现(Computer Implementation)所研究的是计算机系统的物理实现。计算机体系结构、计算机组织、计算机实现三者互不相同但又互相影响。相同体系结构的计算机可以因为速度不同而采用不同的组织结构(如系列机)；同样，一种组织结构可有多种不同的实现方法。

随着计算机体系结构的发展，出现了各种复杂程度、运行速度、处理能力各异的计算机系统，同时也出现了对计算机系统进行分类的不同方法。

1966年，Flynn提出了按指令流和数据流的多倍性对计算机体系结构进行分类的方法。Flynn分类法是基于指令流和数据流这两个概念的，指令流是机器执行的指令序列，数据流是由指令流调用的数据序列，包括输入数据和中间结果。而多倍性是指在系统受限制的部件上，同时处于同一执行阶段的指令或数据的最大数目。从某种程度上说，指令流和数据流是相互独立的，因此有单指令流单数据流(SISD)、单指令流多数据流(SIMD)、多指令流单数据流(MISD)和多指令流多数据流(MIMD)4种类型。

1. SISD 体系结构

符合 SISD 体系结构的计算机代表了传统的冯·诺依曼机器，即大多数单机系统。处理器串行执行指令，或者处理器内部采用指令流水线，以时间重叠技术实现了一定程度上的指令并行执行。但其都是以单一的指令流从存储器取指令，以单一的数据流从存储器取操作数和将结果写回存储器。

2. SIMD 体系结构

SIMD 体系结构有单一的控制部件，但是有多个处理部件。计算机以一个控制单元从存储器取单一的指令流，一条指令同时作用到各个处理单元，控制各个处理单元对来自不同数据流的数据组进行操作。这种体系结构的典型代表是阵列处理机。

3. MISD 体系结构

在 MISD 体系结构中，有几个处理部件，各配有相应的控制部件。各个处理部件接收不同的指令，多条指令同时在一份数据上进行操作。大多数人认为能列在这一系统中的计算机很少或根本不存在。

4. MIMD 体系结构

在 MIMD 体系结构中，同时有多个处理部件，并且每个处理部件都配有相应的控制部件。各个处理部件可以接收不同的指令并对不同的数据流进行操作。大多数现代的并行计算机都属于这一类，多处理机系统和多计算机系统都是 MIMD 型的计算机。

8.1.2 体系结构中的并行性

研究计算机体系结构的目的是提高计算机系统的性能，开发计算机系统的并行性是计算机体系结构的重要研究内容之一。现代计算机的一个共同特点是大量采用并行技术，使计算机的性能得以不断提高。

并行性(parallelism)指的是在同一时刻或是同一时间间隔内完成两种或两种以上性质相同或不相同的工作。也就是说只要时间上互相重叠，就存在并行性。这里，并行性包含同时性和并发性两层含义。其中**同时性**(simultaneity)指两个或多个事件在同一时刻发生的并行性；**并发性**(concurrency)指两个或多个事件在同一时间间隔内发生的并行性。

x86 微处理器的发展就是并行技术发展的一个很好的体现：多流水线、超标量设计等都是提高 CPU 的并行处理能力的关键。而在机群的体系结构中，更是充分利用了并行性这一特点。人们创建和使用并行计算机主要是为了解决单处理器的速度瓶颈，利用并行技术来提高应用性能。

计算机系统中的并行性有不同的等级。不同的分类标准，分类的结果也不相同。

从执行程序的角度看，并行性等级从低到高可分为

(1) 指令内部并行：指的是指令内部的微操作之间的并行。

(2) 指令级并行：指的是并行执行两条或多条指令，就是指令之间的并行。

(3) 线程级并行：指的是并发执行多个线程，通常是以一个进程内控制派生的多个线

程为调度单位的。

（4）任务级或过程级并行：指的是并行执行两个或多个过程或任务（程序段）。

（5）作业或程序级并行：指的是在多个作业或程序间的并行。

不同级的并行性在不同的机器系统中的实现方式不同。在单处理机系统中，这种并行性升到某一级别后（如任务、作业级并行），则要通过软件（如操作系统中的进程管理、作业调度等）来实现。而在多处理机系统中，由于已有了完成各个任务或作业的处理机，其并行性则是由硬件来实现的。

从处理数据的角度，并行性等级从低到高可以分为

（1）字串位串：指的是同时只对一个字的一位进行处理。

（2）字串位并：指的是同时对一个字的全部位进行处理，不同字之间是串行的。

（3）字并位串：指的是同时对许多字的同一位（称位片）进行处理。

（4）全并行：指的是同时对许多字的全部或部分位进行处理。

在一个计算机系统中，可以采取多种并行性措施，既可以有执行程序方面的并行性，又可以有处理数据方面的并行性。

8.1.3　提高并行性的技术途径

1. 三种基本并行技术

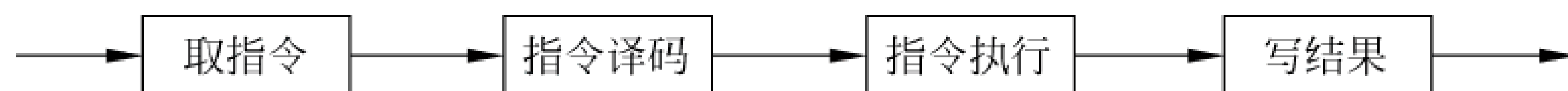
提高并行性的技术途径主要有时间重叠、资源重复和资源共享三种。

1）时间重叠

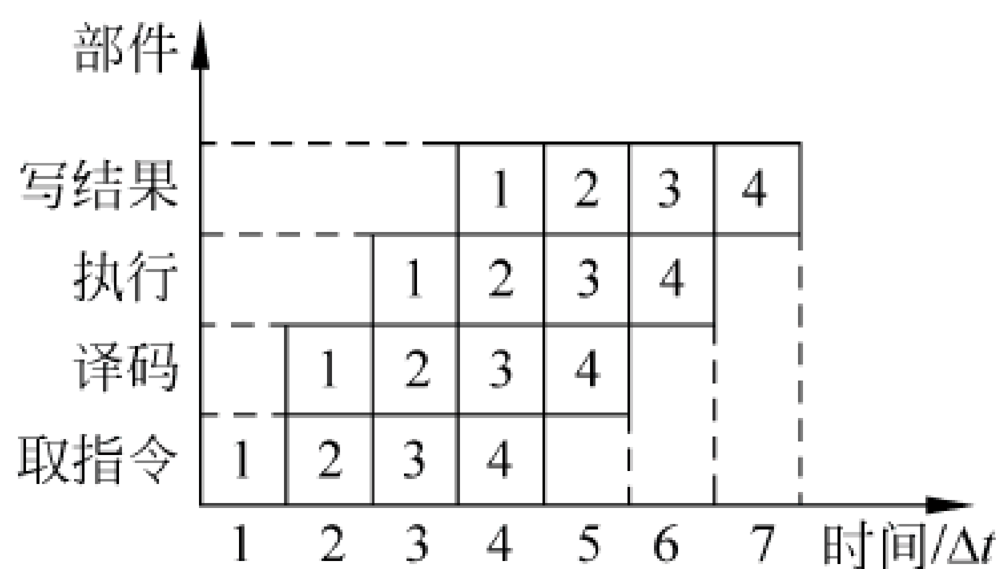
时间重叠指的是多个处理过程在时间上相互错开，轮流重叠地使用同一套硬件设备的各个部分，以加快硬件周转而赢得速度。

实现时间重叠的基础就是部件功能专用化。实质就是把一件工作按功能分割为若干个相互联系的部分；然后把每一部分指定给专门的部件完成；最后按时间重叠原则把各部分执行过程在时间上重叠起来，使所有部件依次分工完成一组同样的工作。流水线技术就是时间重叠的典型应用。

例如，一条指令的执行可以看成由4个过程组成，即取指令、指令译码、指令执行和写结果，如图8-1(a)所示。



(a) 4个子过程的指令流水线



(b) 4条指令流水的时空图

图 8-1　时间重叠的并行技术

将这 4 个过程分别使用 4 个专用的部件实现，即取指令部件（IF）、指令译码部件（ID）、指令执行部件（EX）和写结果部件（WB）。多条指令在时间上相互错开，采用如图 8-1(b)所示的方式工作，轮流重叠地使用同一个部件，加快指令周转来赢得速度。本例中，每个子过程需 Δt 完成，4 条指令如果顺序串行执行需 $4\times4\Delta t=16\Delta t$，而采用时间重叠技术后 4 条指令只需 $7\Delta t$。

时间重叠，因无须重复设置硬件，设备利用充分，因此是单机系统中并行性发展的重要手段。在发展高性能单处理机的过程中，起着主导作用的也是时间重叠这个途径。

2）资源重复

资源重复指的是根据“以数量取胜”的原则来实现并行，其付出的代价是在空间上通过重复地设置资源，尤其是硬件资源，以提高计算机系统的性能。

例如，图 8-2 中，设置了 n 个相同的处理单元 PE_1、PE_2、…、PE_n，在同一个控制单元（CU）的控制下，给各处理单元分配不同的数据完成同一种运算或操作，以提高处理速度。

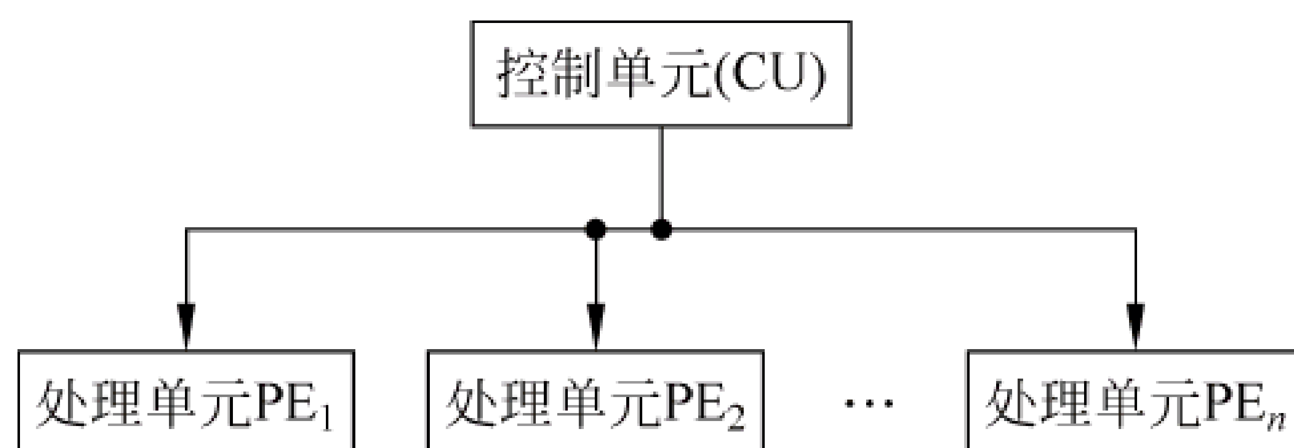


图 8-2　资源重复的并行

资源重复也是计算机系统中经常采用的并行技术，通过使用各种资源重复技术，如部件冗余、多操作部件、多存储体、并行处理机、相联处理机、多处理机系统等，可以有效提高计算机系统性能。

3）资源共享

资源共享是一种软件方法的并行，它使多个任务按一定的时间顺序轮流使用同一套硬件设备。资源共享的实质就是用单处理机模拟多处理机的功能，形成所谓虚拟机的概念，体现在：多道程序、分时系统、多终端、远程终端、智能终端、分布处理系统（把若干个具有独立功能的处理机或计算机相互连接起来，在操作系统的全盘控制下，统一协调地工作，而最少依赖集中的程序、数据或硬件）等方面的应用上。

2. 多机系统的并行性

多机系统包括多处理机系统和多计算机系统。多机系统也遵循时间重叠、资源重复和资源共享这三种基本的技术途径，向着三种不同的多处理机方向发展。但在采取的技术措施上与单机系统有所不同。

通常用耦合度来反映多机系统的各机器之间的物理连接的紧密程度和交互作用能力的强弱。多机系统的耦合度分为最低耦合、松散耦合、紧密耦合等几类。

1）最低耦合

耦合度最低的系统。除通过某种中间存储介质之外，各计算机之间没有物理连接，也无共享的连机硬件资源。

2）松散耦合或间接耦合系统

松耦合系统的各处理机间通过共享 I/O 子系统、通道或通信线路实现处理机间的通信和互连，不共享主存，但可共享某些外围设备（如磁盘、磁带等），机间的相互作用是在文件或数据集一级进行的。松散耦合多处理机由多个处理机、一个通道、一个仲裁开关和消息传送系统组成。每个处理机带有一个局部存储器和一组 I/O 设备。在仲裁开关的通道中有高速通信存储，用来缓冲传送的信息块。

3）紧密耦合系统或直接耦合系统

紧耦合多处理机系统，其处理机间物理连接的频带较高，它们往往通过总线或高速开关实现互连，可以共享主存，各处理机之间是通过互联网络共享主存的。一般地，紧耦合系统由 P 台处理机、m 个存储器模块、d 个 I/O 通道和三个互联网络构成。其中，处理机-存储器网络实现处理机与各存储模块的连接；处理机中断信号网络实现多处理机之间的互连；处理机-I/O 互联网络实现处理机与外设的连接。每个处理机可自带局部存储器，也可自带 cache 存储器模块，可采用流水工作方式。紧耦合系统多用于并行作业中的多任务，一般处理机是同构的。

8.2　流水线技术

计算机中的流水线技术来源于工业领域的流水线生产，属于并行处理技术中的一种时间重叠。流水线技术最早在大型计算机系统中采用，而现代计算机无论是小型还是微型计算机均在 CPU 中大量使用了该技术，以提高 CPU 执行指令的速度。

8.2.1　流水线的基本概念

在 4.3 节曾经提到了流水线技术。每当提起流水线技术，人们就容易将其与工业生产中的产品生产流水线联系起来。的确，计算机中的流水线概念正是由此得来的。为了对计算机中的流水线概念有更深入的认识，首先来看看产品生产流水线的情况。

假设在某工厂车间，生产某个产品需要 4 道工序，该产品生产车间以前只有一个工人，一套生产该产品的设备。该工人每天工作 8 小时，每 4 分钟可以生产该产品 1 件，这样一天可以生产该产品 120 件。现在工厂希望将该产品的日产量提高到 480 件，那么采取什么办法才能实现这一目标呢？一种很容易想到的办法就是增加 3 个工人和 3 套设备，采用传统的生产方式，即每个工人单独使用一套设备进行生产，每个工人日产量是 120 件，4 个工人就是 480 件。这种方法当然是可行的，但其付出的代价是增加了 3 套设备。另一种办法就是只增加 3 个工人，不增加设备。同时对设备的生产流程进行改造：将 4 道工序分离开来，使得每道工序的生产时间一样（均为 1 分钟），每个工人专做一道工序，每完成一道工序就将半成品交给下一道工序的工人，直至生产出完整的产品，如图 8-3 所示。经过这种改造以后，每隔 1 分钟就有 1 件产品生产出来，这样的话，一天 8 小时就同样可以生产 480 件该产品了。后一种方法与前一种方法相比，节省了 3 套设备，而实现的效果是相同的，具有更好的“性能价格比”。

在早期的计算机系统中，无论是指令的执行还是数据的处理，都是严格地按照串行的方

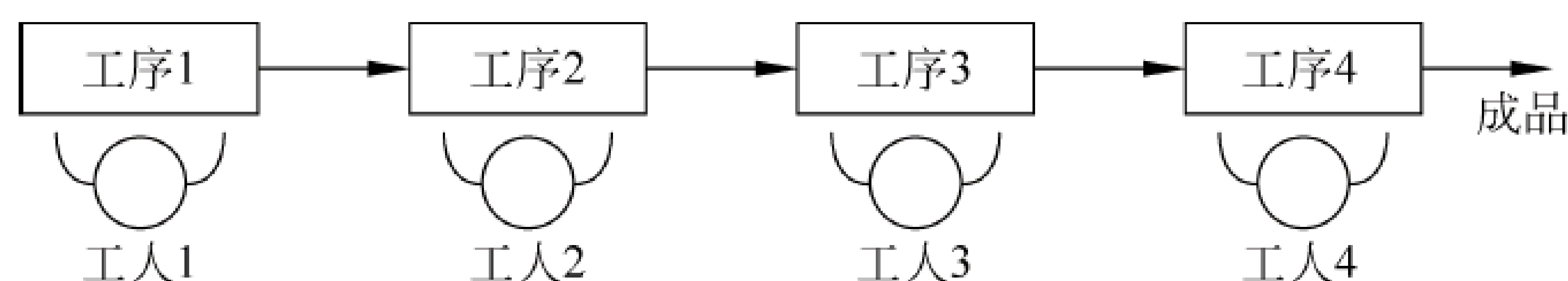


图 8-3　产品生产流水线

式来进行的，也就是指令或数据流入某个部件后，无论需要执行或处理的时间是长还是短，这时除了正在执行或处理的部件以外，其余的部件都处于闲置状态。这样的处理方式控制简单，但效率低下，因为这些闲置的部件完全可以在等待的时间里完成一定的工作。

将工业生产中的流水线思想引入计算机中。例如，可以将计算机中的指令部件划分成两个子部件：取指令部件和执行部件。取指令部件专门负责从存储器中取指令；而执行部件专门负责完成指令的译码和执行等操作。现假设这两个部件完成一次操作所需时间均为 Δt，按传统的串行方式工作的话，该指令部件完成一条指令从取指到执行的时间为 $2\Delta t$；而采用流水线工作方式的话，则每隔 Δt 的时间就可以完成一条指令的执行。显然，指令执行速度提高了一倍。再如，对于浮点加法器而言，可以把浮点加法的全过程分解为求阶差、对阶、尾数加和规格化 4 个子过程，让每个子过程都在各自独立的部件上完成。同样假设这 4 个部件每个完成一次操作的时间均为 Δt，按传统的串行方式工作的话，该浮点加法部件完成一次浮点加的时间为 $4\Delta t$；而采用流水线工作方式的话，则每隔 Δt 的时间就可以完成一次浮点加。显然，一次浮点加的速度提高了 3 倍。

实际上，计算机中的流水线技术是一种利用时间重叠技术提高机器性能的并行处理技术，它能在不增加机器硬部件的情况下，通过对某一部件功能进行合理的分解与设计，有效提高部件的处理速度。目前流水线技术大量应用在 CPU 内部，包括组成 CPU 各部件之间的流水和部件内部的流水等。流水线技术的应用对于提高处理机的性能起到了相当大的作用。

8.2.2　流水线的分类

流水线可以从不同的角度进行分类，一般来讲流水线可以分为以下几种类型。

1. 部件级、指令级和处理机级流水线

按照计算机处理的级别来分类，流水线可以分为部件级流水线、指令级流水线和处理机级流水线。

部件级流水指的是构成部件内的各个子部件间的流水，如运算器内浮点加、减运算的流水。指令级流水指的是构成计算机系统的多个处理机之间的流水，又称为宏流水。处理机级流水指的是构成处理机的各部件之间的流水，如一条指令分成“取指”、“分析”、“执行”后，这三者之间的流水。

2. 单功能流水线和多功能流水线

按照流水线可以完成的动作数量来分类，又可以分为单功能流水线和多功能流水线。单功能流水线指的是只能实现单一功能的流水，如只能实现浮点加减而不能实现浮点乘除的流水线。当然，如果要完成多种功能的流水可以将多个单功能的流水线组合起来实现。

多功能流水线指的是在同一流水线的各个段之间可以用多种不同的连接方式以实现多种不同的功能或运算。

如图 8-4 所示的就是同一个流水线在不同连接下可以分别实现浮点加、减运算时的连接和定点乘、除法运算时的连接。

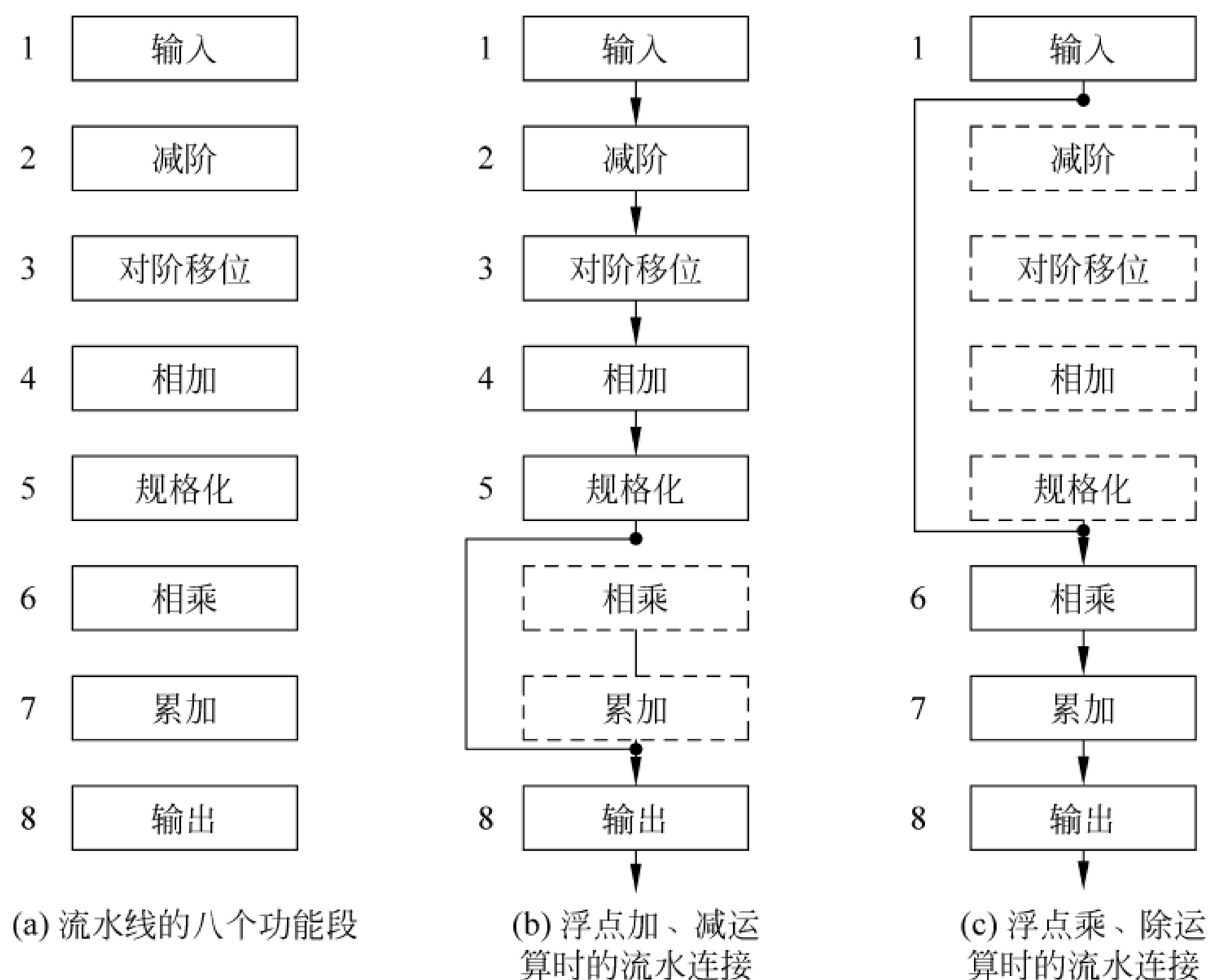


图 8-4 多功能流水线的实现

3. 静态流水线和动态流水线

按照流水线的各个段是否允许同时进行多种不同功能的连接流水，可以把流水又分成静态流水线和动态流水线。

静态流水线指的是在某一段时间内各功能段只能按照一种功能连接形成流水线，只能等待该流水线上全部任务流空后，才能切换成另一种功能连接进行流水。显然，这种流水操作，实现起来很简单，所需的控制方式也不复杂。但就指令级的流水而言，如果进入的是一串相同的运算指令，静态流水性能还不错；但如果进入的是浮点加、定点乘、浮点加、定点乘、……这样一连串功能相异的指令时，静态流水线的性能就降低得还不如指令顺序执行方式。而动态流水线是指在同一时间内，当某些段正在实现某种运算时，另一些段却可以实现另一种运算。这样，就不是非得相同运算的指令才能进入流水处理。显然，这对提高流水线的效率很有好处。然而，这却会使流水线的控制变得很复杂。

例如，假设先后有两批任务要完成，第一批是 n 个任务的浮点加、减运算；第二批任务是 A～E 共 5 个任务的定点乘法运算，分别按照静态、动态两种不同的方式形成流水线，其流水线时空图如图 8-5 所示。

从软硬件功能分配的观点上看，静态流水线是功能负担较多地加在软件上了，以简化硬件控制；而动态流水线则是将功能更多地由硬件来实现，以提高流水的效能。目前大多数高性能流水计算机采用多功能静态流水线，就是因为其控制和实现都比较简单。

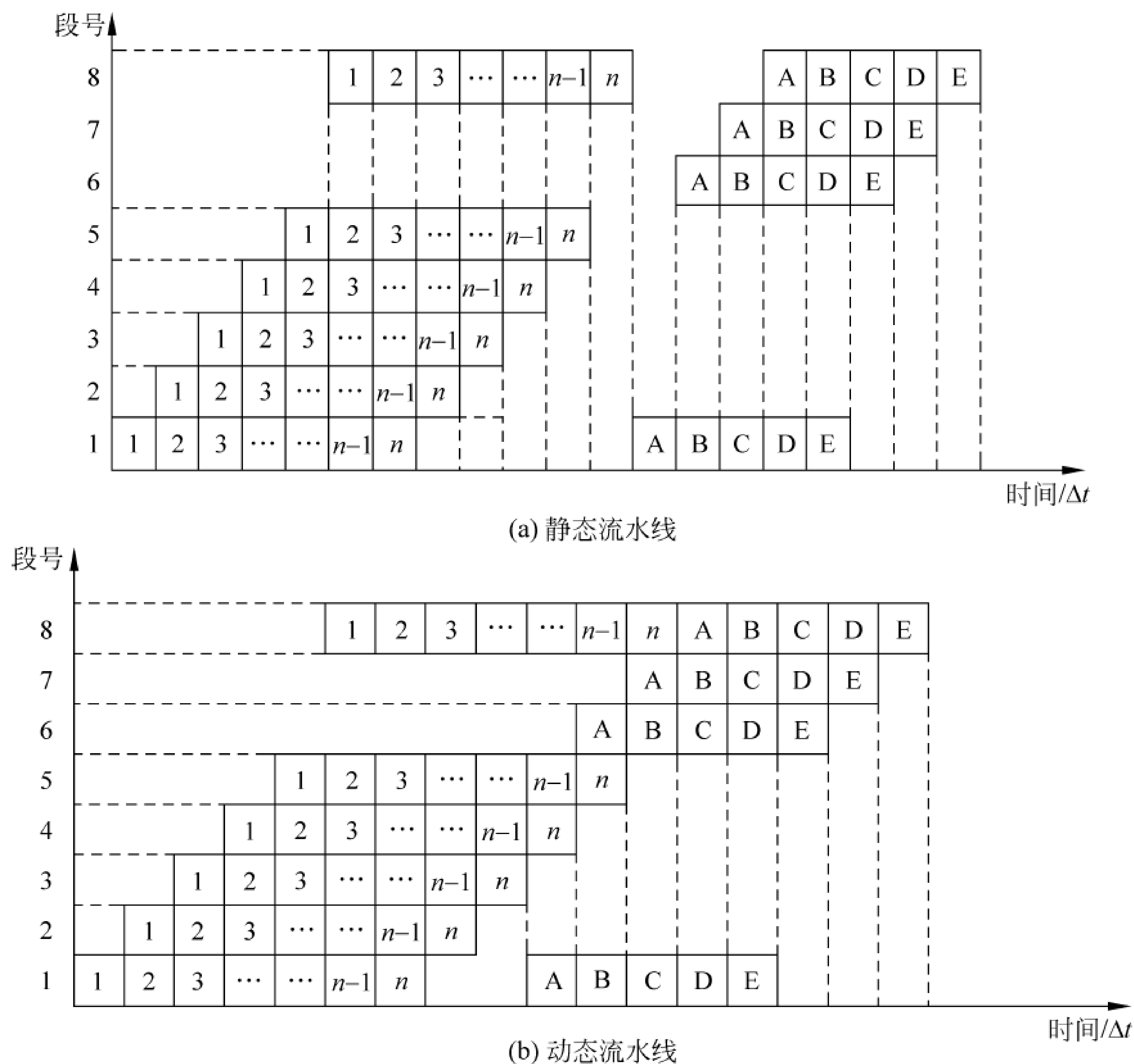


图 8-5 静态、动态多功能流水线时空图

4. 线性流水线和非线性流水线

如果按照流水线内部功能部件的连接方式(如各功能段之间是否有反馈回路)来分类,可分为线性流水线和非线性流水线。

流水线各段之间串行连接,各段只经过一次,且各功能段之间没有反馈回路的,就称为线性流水线。反之,如果流水线除了有串行连接的通路,还有反馈回路,并且往往使得任务流经流水线时需多次经过某个段或越过某些段的,就称为非线性流水线,如图 8-6 所示。

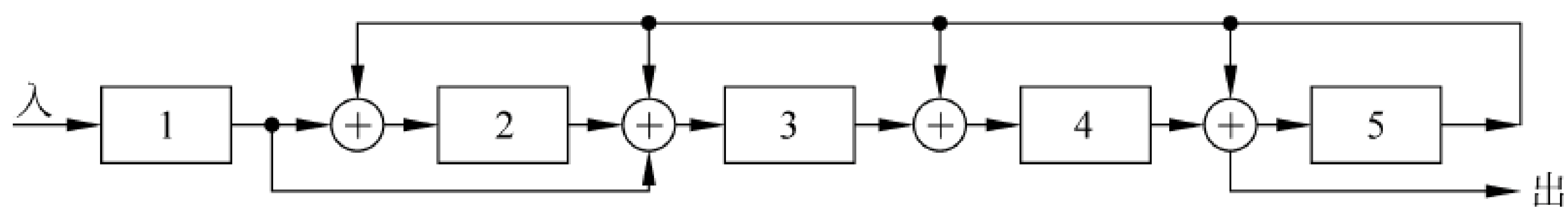


图 8-6 非线性流水线举例

5. 标量流水线和向量流水线

如果按照机器可处理的对象来分类,还可以分为标量流水线和向量流水线。

按照机器可处理的对象来分类实质就是从机器所具有的数据表示来分类,此时一般也

意味着针对该机器所具有的这些数据表示会获得相应的硬件支持。所以标量流水机意味着该机器没有向量数据表示，只能用标量循环方式来处理向量和数组。向量流水机则意味着该机器有向量数据表示，并且硬件上有相应的部件的支持，如机器上一般都会设置有向量指令和向量运算硬件，能流水地处理向量和数组中的各个元素。显然，向量流水机是向量数据表示和流水技术相结合的产物。

8.2.3 流水线的主要性能参数

衡量一种流水线处理方式的性能高低的参数主要有吞吐量、加速比和效率。

吞吐量(Throughput，Tp)，指的是计算机中的流水线在单位时间内能流出的任务数或结果数。流水线的吞吐量可以进一步分为最大吞吐量和实际吞吐量。它们主要和流水段的处理时间、缓存寄存器的延迟时间有关，流水段的处理时间越长，缓存寄存器的延迟时间越大，那么，这条流水线的吞吐量就越小。因为，在线性流水线中，最大吞吐量 $Tp_{max}=1/\Delta T=1/\max(\Delta T_1,\cdots,\Delta T_i,\cdots,\Delta T_m)$，其中，$m$ 是流水线的段数，ΔT_i 表示的是第 i 段的执行时间。显然，最大吞吐量受到了瓶颈段的约束，这里，瓶颈段指的是所有段中执行时间最长的那个段。为了提高流水线的最大吞吐量，就要找到瓶颈段，并设法消除此瓶颈。

例如，某流水线有 4 个段，其中 2 号段由于需用时 $3\Delta t$，所以 2 号段是瓶颈段，见图 8-7(a)；5 个任务流经该流水线时的流水效果的实际情况如图 8-7(b)所示，显然流水速度受到了 2 号瓶颈的限制。

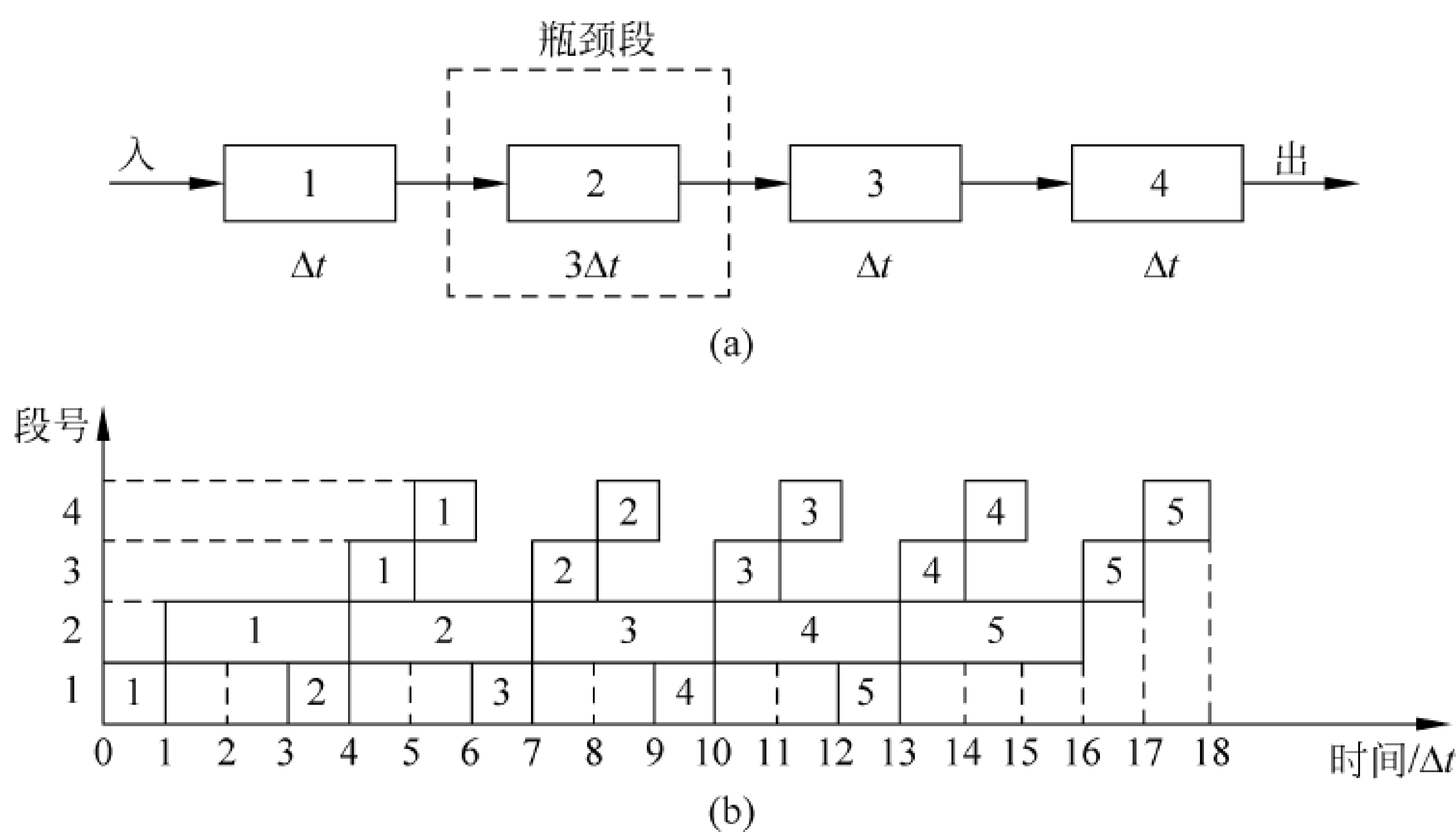
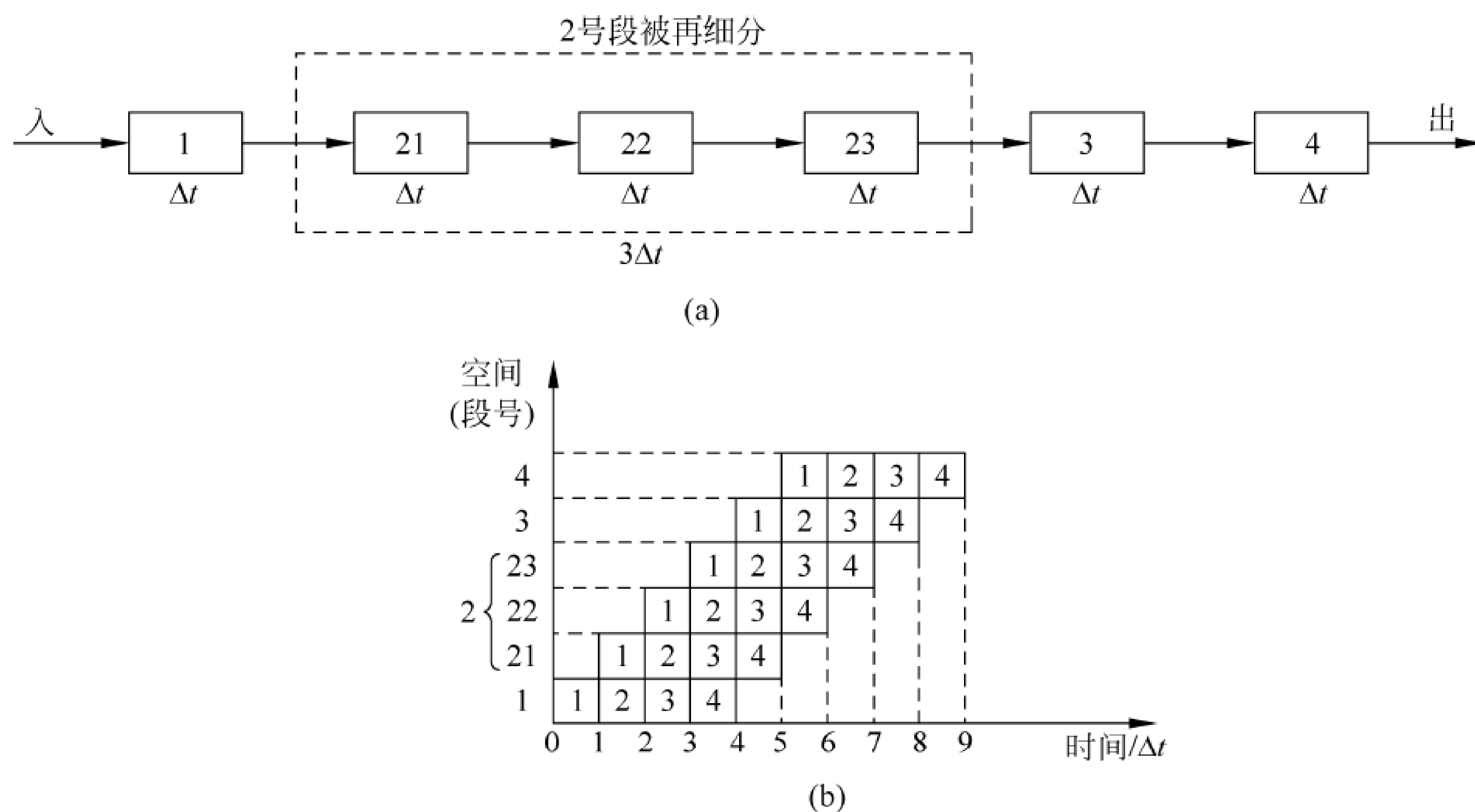


图 8-7 最大吞吐量取决于瓶颈段的时间

消除瓶颈部分对流水线吞吐量的影响一般有两种方法：

(1) 把瓶颈部分的流水线分拆，以便任务可以充分流水处理。

流水段的处理时间过长，一般是由于任务堵塞造成的，而任务的堵塞会导致流水线不能在同一个时钟周期内启动另一个操作，那么，可以将流水段进行划分，在各小流水段中间设置缓存寄存器，缓冲上一个流水段的任务，使流水线充分流水。例如，某流水线，除 2 号流水段外，各功能段处理时间都是 Δt，假如 2 号流水段的处理时间为 $3\Delta t$，那么，可以把 2 号流水段再细分成三小段，这样，每小段的功能相同，但是处理时间已经变成 $3\Delta t/3=\Delta t$ 了，如图 8-8 所示。



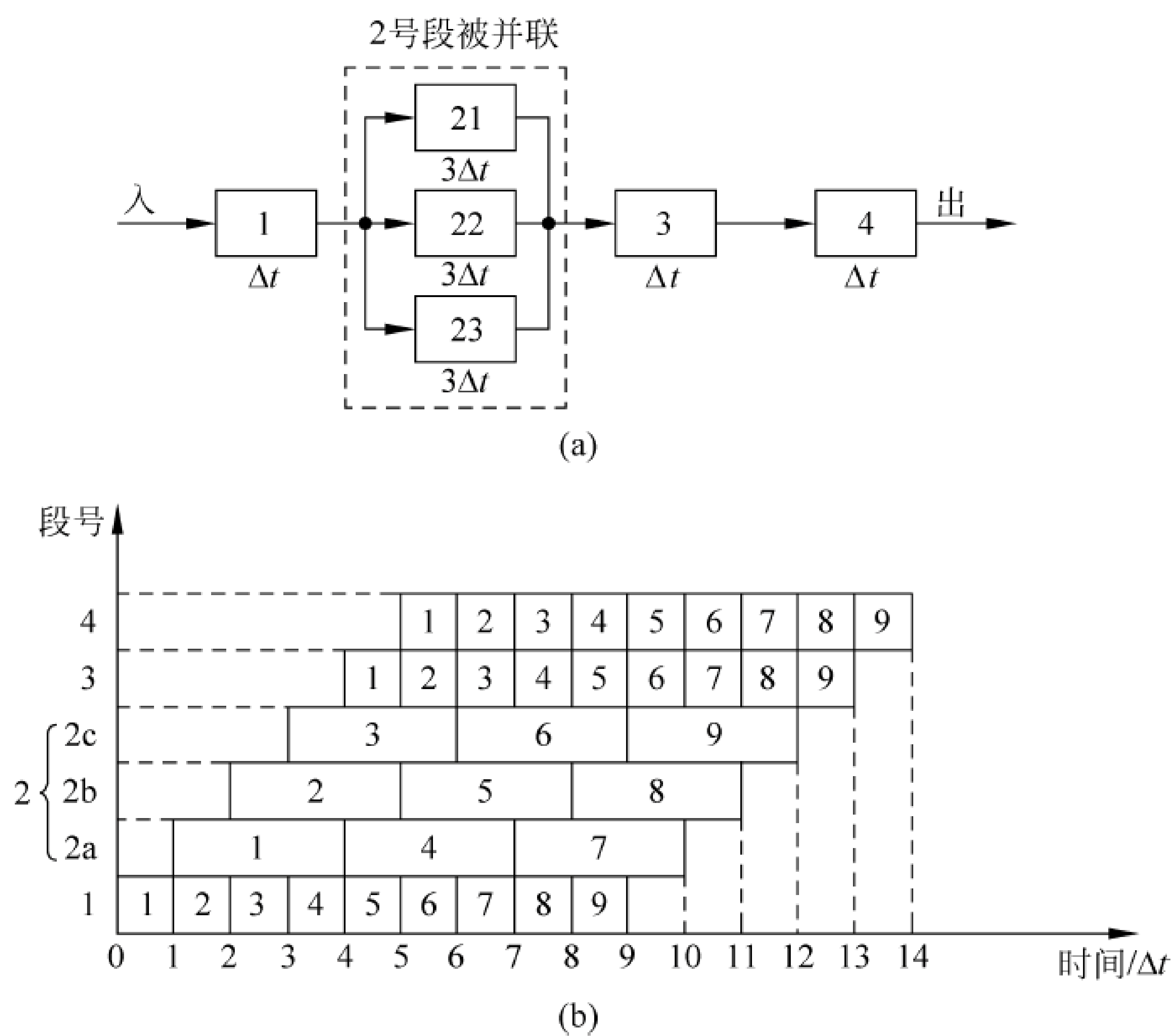
(a)

(b)

图 8-8　瓶颈子过程再细分

（2）在瓶颈部分设置多条相同的流水段，并行处理。

上例中，对付 2 号流水段的处理时间过长，还有另外一种方法，那就是把瓶颈流水段用多个相同的并联流水段代替，在前面设一个分派单元来对各条流水段的任务进行分派。仍然假设瓶颈流水段的处理时间是 $3\Delta t$，那么经过 3 条并联流水段的同时处理，实际需要的时间只是 Δt。这样，就达到了缩短流水段的处理时间的目的，但这种方法因为要三段相同的流水段并联，设备成本较高，而且，需要解决好在各并行子过程之间的任务分配和同步控制问题，比瓶颈子过程再细分的控制要复杂得多，如图 8-9 所示。



(a)

(b)

图 8-9　瓶颈子过程并联

下面介绍流水线的实际效率的分析并计算。

由于流水线在开始工作时存在建立时间，在结束时存在排空时间，加上各种原因使流水线不能连续流动，流水会出现断断续续的现象，所以流水线的实际吞吐量 Tp 总比最大吞吐量 Tp_{max} 要小。图 8-10 是一条有 4 个段的静态流水线，假设每段所需时间为 Δt，n 个任务全部进入流水线所需的时间 $T_0=n\Delta t$，流水线建立所需时间（就是一个任务全部进入流水线所需时间）为 $T_1=m\Delta t$，之后 $n-1$ 个任务流出流水线需时间 $T_2=(n-1)\Delta t$，则 n 个任务完成全部从流水线流空所需的时间为 $T=T_1+T_2=m\Delta t+(n-1)\Delta t$，$n$ 个任务的实际吞吐量 $\mathrm{Tp}=n/(T_1+T_2)=n/(m\Delta t+(n-1)\Delta t)$。

具体地，仍然见图 8-10，相隔 ΔT 时间后，有 5 个任务（$n=5$）A、B、C、D、E 进入了流水线，这 5 个任务的流水中，$T_0=n\Delta t=5\Delta t$，$T_1=m\Delta t=4\Delta t$，$T_2=(n-1)\Delta t=4\Delta t$，$T=T_1+T_2=m\Delta t+(n-1)\Delta t=4\Delta t+(5-1)\Delta t=8\Delta t$，实际吞吐量 $\mathrm{Tp}=5/(8\Delta t)$。

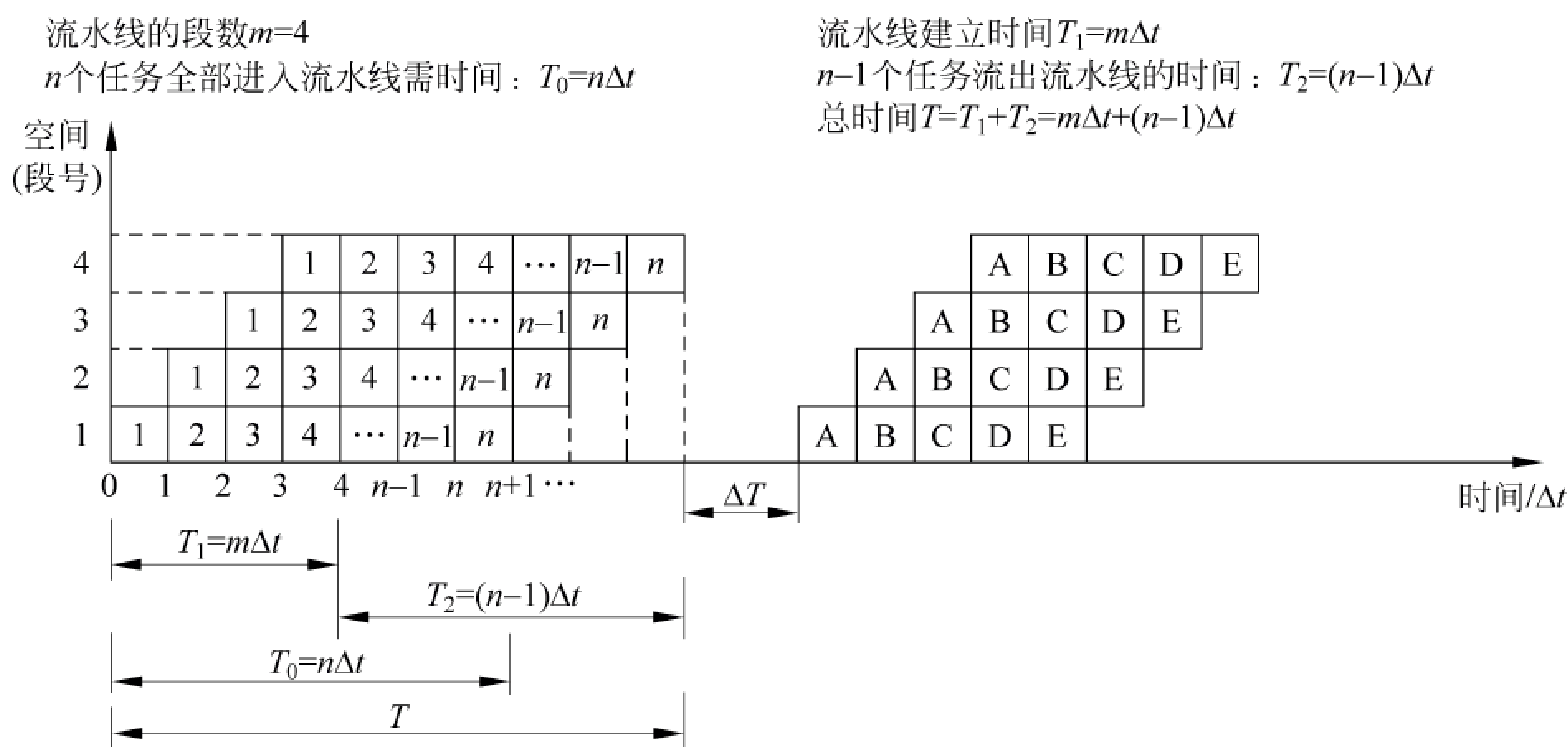


图 8-10　分析实际吞吐量的时空图

显然，如果一条流水线的段数越多，子过程执行时间越长，那么，这条流水线的理论吞吐量就越小。

加速比（Speed Ratio）：表示流水方式相对于非流水顺序方式速度提高的比值，具体就是指某一流水线如果采用串行模式之后的工作速度和采用流水线模式后的工作速度的比值。可以看出，数值越大，说明这条流水线的工作安排方式越好。假设该流水线各子功能段执行时间均为 Δt，流水段有 m 个段，那么 n 个任务非流水顺序完成需要 $n\times m\times\Delta t$ 的时间；流水完成则需要 $m\Delta t+(n-1)\Delta t$ 的时间，因此，流水方式工作的加速比为

$$\mathrm{Sp}=\frac{n\cdot m\cdot\Delta t}{m\Delta t+(n-1)\Delta t}=\frac{m}{1+\dfrac{m-1}{n}}$$

所以在流水线各子功能段执行时间均相等的情况下，仅当 $n\gg m$ 时，其加速比才能趋近于最大值 m，即流水线的段数。

效率（Efficiency）：也称流水线设备的时间利用率，又称使用效率，它指的是流水线中各个部件的利用率，也就是设备的实际使用时间占整个运行时间的比值。显然，效率总是小于 100％的，这是由于流水线在开始工作时存在建立时间；在结束时存在排空时间，各个部件

不可能一直在工作，总有某个部件在某一个时间处于闲置状态，这同时也意味着可以用处于工作状态的部件数量和总部件数量的比值来粗略地说明这条流水线的工作效率。

如果是线性流水线，任务间不相关且各段经过的时间相同，那么以图 8-10 为例，在 T 时间里流水线的各段效率都相同，均为 η_0，整个流水线的效率就是 η。

$$\eta_1 = \eta_2 = \cdots \eta_m = \frac{n \cdot \Delta t}{T} = \frac{n}{m+n-1} = \eta_0$$

$$\eta = \frac{\eta_1 + \eta_2 + \cdots + \eta_m}{m} = \frac{m \cdot \Delta \eta_0}{m} = \frac{m \cdot n\Delta t}{m \cdot T}$$

以上 η 的计算公式中，分母 $m \cdot T$ 正好是图 8-10 中 m 个段和流水总时间 T 所围成的面积，分子 $m \cdot n \cdot \Delta t_0$ 则是图 8-10 时空图中 n 个任务实际使用的面积。这就是说，效率实际上就是 n 个任务占用的时空区面积与 m 个段总的时空区面积的比值。显然，只有当 $n \gg m$ 时，η 才趋近于 1。

8.2.4 流水线的相关问题

流水线只有连续不断地流动，不出现断流，才能获得高效率。如果处理不当，使流水线产生“断流”，将会使流水效率显著下降。流水过程中因为相关问题而产生冲突，是导致流水线断流的主要原因，一般来讲，流水线的相关主要分为以下三种类型。

1. 结构相关

结构相关是指当指令在重叠执行过程中，硬件资源满足不了指令重叠执行的要求，两条或两条以上指令争用同一资源而引起的冲突，因此，结构相关又称为资源相关。

例如，假设一条指令流水线由 5 段组成，分别为取指令（IF）、指令译码（ID）、取操作数（MEM）、执行运算（EX）和写寄存器（WR）。该流水线的时空图如图 8-11 所示。

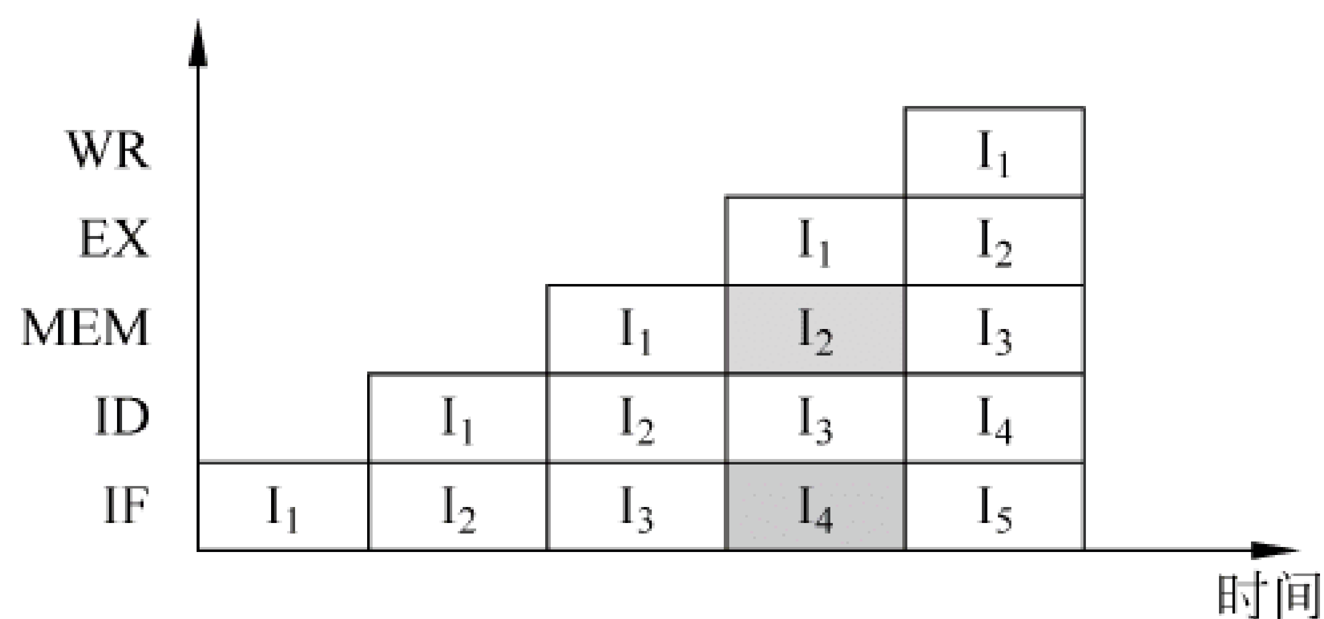


图 8-11 5 段指令流水线

从图 8-11 中可以看出，指令 I_2 的取操作数和指令 I_4 的取指令都需要访问存储器。若机器中只有一个单端口存储模块，那么 I_2 的取操作数和指令 I_4 的取指令就产生了访存冲突，两个操作无法同时进行，这就是一种典型的资源冲突。

一种解决这种冲突的方法是在机器中增加存储器模块，如使用双端口存储器，使指令和数据分别存放在不同的存储器模块中，这样，取操作数和取指令就不会发生冲突。另一种方法是，当发生取操作数或取指令冲突时，将其中一个操作的执行时间推迟，如图 8-12 所示。当然，这样的话也就是发生了流水线的断流，流水线的吞吐量将会下降。

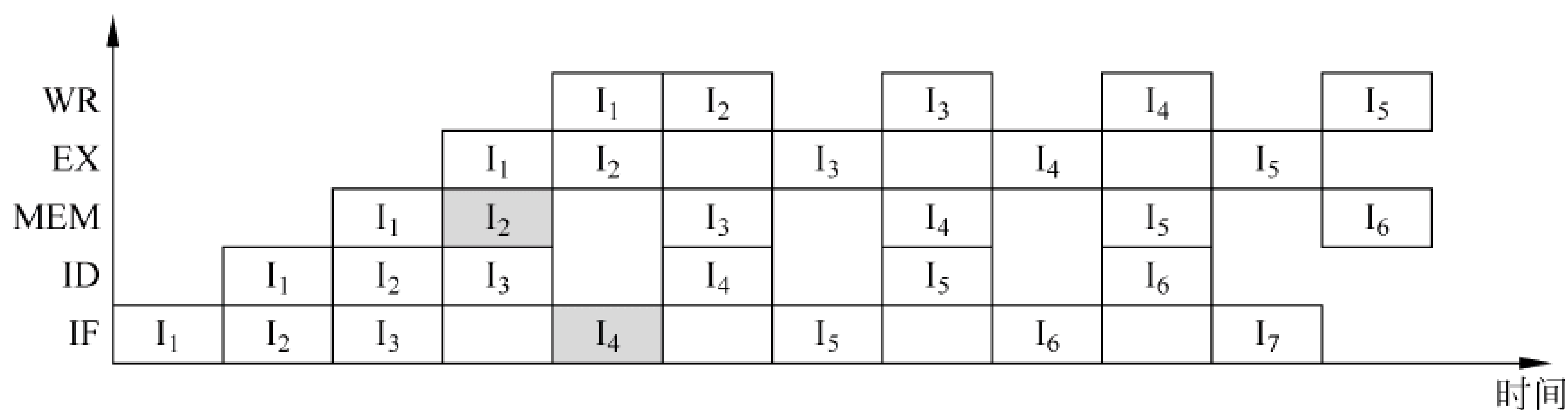


图 8-12 访存相关引起流水线断流

2. 数据相关

当一条指令需要用到前面指令的执行结果,而这些指令均在流水线中重叠执行时,就有可能产生数据相关。

在流水计算机中,指令的处理是重叠进行的,前一条指令还没有结束,第二、第三条指令就陆续地开始工作。由于多条指令的重叠处理,当后继指令所需的操作数,刚好是前一指令的运算结果时,便发生数据相关冲突。例如,某一时间以下三条指令在图 8-12 的流水线中执行。

$$\begin{array}{ll} \text{ADD} & R_1, R_2, R_3; (R_2)+(R_3)\rightarrow R_1 \\ \text{SUB} & R_4, R_1, R_5; (R_1)-(R_5)\rightarrow R_4 \\ \text{AND} & R_6, R_1, R_7; (R_1)\wedge(R_7)\rightarrow R_6 \end{array}$$

其中,SUB 指令的 EX 段需要执行 R_1 减 R_5,而同一时间,其上一条指令正在执行写结果到 R_1 的操作。这种情况下,有可能 SUB 指令的 R_1 并不是 ADD 指令最终形成的 R_1 结果,则程序的执行将发生错误。

数据相关包括的情况很多,如主存操作数相关、通用寄存器组的数相关及基址值或变址值相关等,它们都属于局部性相关,都是由于在机器同时解释多条指令时出现了对同一主存单元或寄存器要求"先写后读"的冲突而造成的。解决这种相关的一种方法是推后后续指令对相关单元的读,保证在先的指令写入先完成。例如,对于上面的例子,为了保证上述指令序列执行的正确性,流水线只能暂停 SUB 指令的执行,直到 ADD 指令将结果写入 R_1 寄存器后,再启动 SUB 指令的执行。

另外,任务在流水线中的流动顺序的安排和控制可以有两种方式。一种是任务流入和流出的顺序一致,称为顺序流动方式或同步流动方式;另一种是任务流出和流入的顺序可以不同,称为异步流动方式。

例如,有一个 8 段的流水线,其中第 2 段是读段,第 7 段是写段,如图 8-13 所示。一串指令 h、i、j、k、l、m、n、…依次流入,当指令 j 的源操作数地址与指令 h 的目的操作数地址相同时,h 和 j 就发生了"先写后读"的数相关,解决办法如图 8-13 所示,设置"相关直接通路",让 h 的结果直接写入 j 所要访问的存储单元,这样就可以保证 j 读到的操作数是 h 写入的结果;此外,当流水线采用异步流动方式时,会出现很多顺序流动不会发生的其他相关。比如,若指令 i、k 都有写操作且写入同一单元时,可能由于指令 i 执行时间长或有"先写后读"相关,就会出现指令 k 先于指令 i 到达"写",结果错误地使得该单元的最后内容为指令 i 的写入结果。这种要求在先的指令先写入,在后的指令才写入的相关为"写-写相关"。此外,

当指令 i 的读操作和指令 k 的写操作是对同一单元时，若由于指令 k 越过指令 i 向前流，且它的写操作先于指令 i 的读操作开始前完成，那指令 i 就会读到错误的数据。这种对同一单元要求在先的指令先读出，在后的指令才写入的相关称为“先读后写”相关。对于这两类相关，只有异步流动时才可能发生，同步（即顺序）流动是不可能发生的，解决它们的办法就是在控制机构上要保证写入与写入之间、读出与写入之间的先后顺序不变。

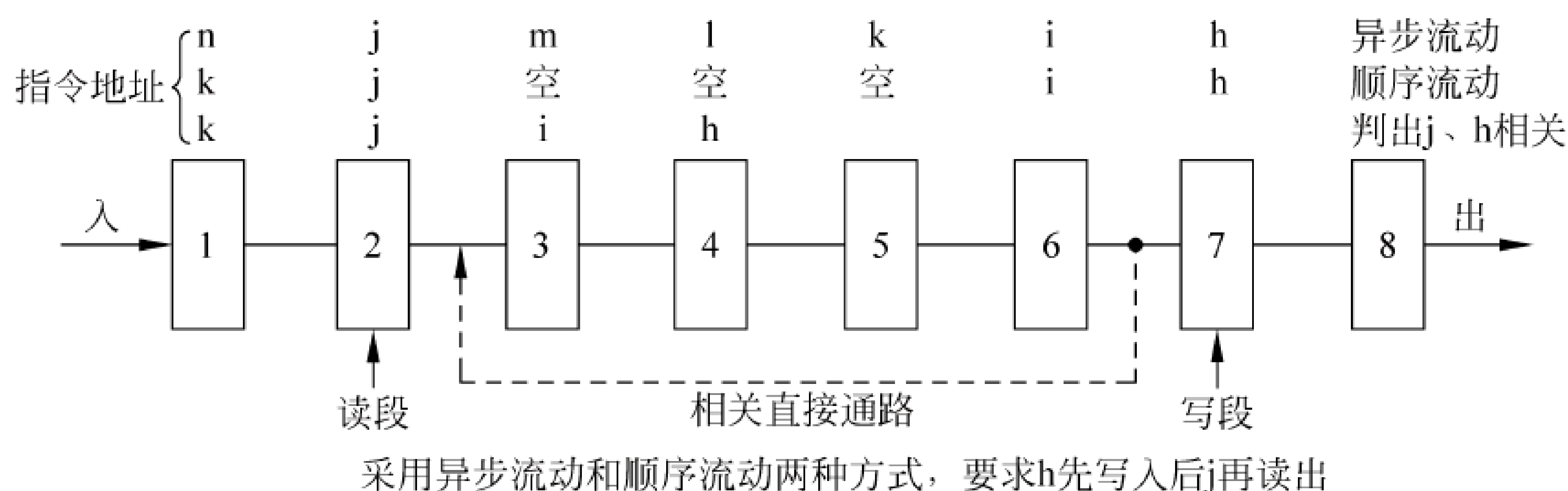


图 8-13 顺序流动和异步流动

3. 控制相关

控制相关冲突是由转移指令引起的。当执行转移指令时，依据转移条件的产生结果，可能为顺序取下条指令；也可能转移到新的目标地址取指令，从而使流水线发生断流。

为了减小转移指令对流水线性能的影响，常见的解决办法如下。

1）猜测法

顾名思义，猜测法就是当遇见转移指令 i 时，会形成两个分支，i+1、i+2、…，是转移不成功时继续执行的一路分支，另一路分支是转移成功时转向执行指令 p、p+1、…。流水意味着同时解释多条指令，i 进入流水线后，后面到底是执行 i+1 还是 p 指令要等指令 i 的条件码建立以后才知道，而 i 的条件码建立一般要等到条件转移指令 i 快流出流水线时才行，那么在没有建立 i 的条件码之前，i 之后的指令停顿下来的话，流水就断流了，性能肯定下降。为了不断流，可采用猜测法猜取 i+1 和 p 两个分支中的一个继续向前流动。

可是，猜哪个分支好呢？如果程序中这两个分支被执行的概率相近的话，应首选 i+1 分支，因为至少它已经取进了指缓里了，可以很快从中取出指令 i+1 进入流水线而不必等待。反之，因为 p 可能尚未取进指缓，需花时间访存取出，导致流水断流。如果两个分支被执行的概率不均等的话，当然还是选高概率的分支。那么怎样判断两者谁是高概率的分支呢？一种办法是静态地根据转移指令类型或程序执行期间转移的历史状况来预测，这种静态策略意味着需要事先对大量程序的转移类型和转移概率进行统计，且不一定能保证有较高的猜测准确度。也可以采用动态策略，就是由编译程序根据执行过程中转移的历史记录来动态地预测未来可能的转移选择。

问题是，不管怎么猜选，如果猜对了倒好，可要是猜错了怎么办呢？当然是尽快回到原分支点去转入执行另一分支。就这么简单吗？当然不是，因为猜错了意味着分支点原先的现场很可能已被修改了。所以不管怎么猜选，都要能保证猜错时可恢复原先的现场信息。解决的办法是设置后援寄存器，把那些可能被破坏的原始状态信息都用后援寄存器保存起来，一旦猜错就取出后援寄存器的内容来恢复分支点的现场。

2）加快和提前形成条件码

尽快尽早地获得条件码，就可以提前知道流水线将流向哪个分支。

① 加快单条指令内部的条件码的形成，尤其是某些反映运算结果的条件码完全可以不必等到指令执行完就提前形成。比如，根据运算规律来看，乘、除运算的结果是正是负的条件码就完全可以在运算前形成。

② 在一段程序内提前形成条件码，比如循环程序，一般是根据循环条件判断是否继续转移。很多循环程序的循环次数虽然是在循环末端语句对循环次数减 1，然后再判断循环次数是否为 0 来决定是否结束循环的。细想一下不难发现，循环次数减 1 和判断循环次数是否为 0 的操作，完全可以提前完成，甚至可以提前到循环体开始时就进行。

3）采用延迟转移

采用延迟转移办法是用软件方法进行静态指令调度的技术，就是在编译生成目标指令程序时，将条件转移指令与它前面不相关的一条或多条指令交换位置，让成功转移总是延迟到在这一条或多条指令执行之后再进行。延迟转移方法因为思路简单，而且不必增加硬件，故比较实用。

4）加快短循环程序的处理

① 为避免短循环程序取进了指缓后，由于指令预取导致指缓中需循环执行的指令被冲掉，为减少访存次数，可将短循环程序一次性整个地装入指缓内，以加快短循环程序的处理。

② 由于循环分支概率高，让循环出口端的条件转移指令恒猜循环分支，就可以降低因为条件分支而造成的流水线断流的概率。

8.2.5　超流水线技术

1. 超标量处理机

假设一条指令包含取指令、译码、执行和存结果 4 个子过程，每个子过程经过的时间为 Δt。常规标量单流水处理机是在每个 Δt 期间解释完一条指令，如图 8-14 所示。完成 9 个任务需要 $12\Delta t$ 的时间，称这种流水机的度 $m=1$。

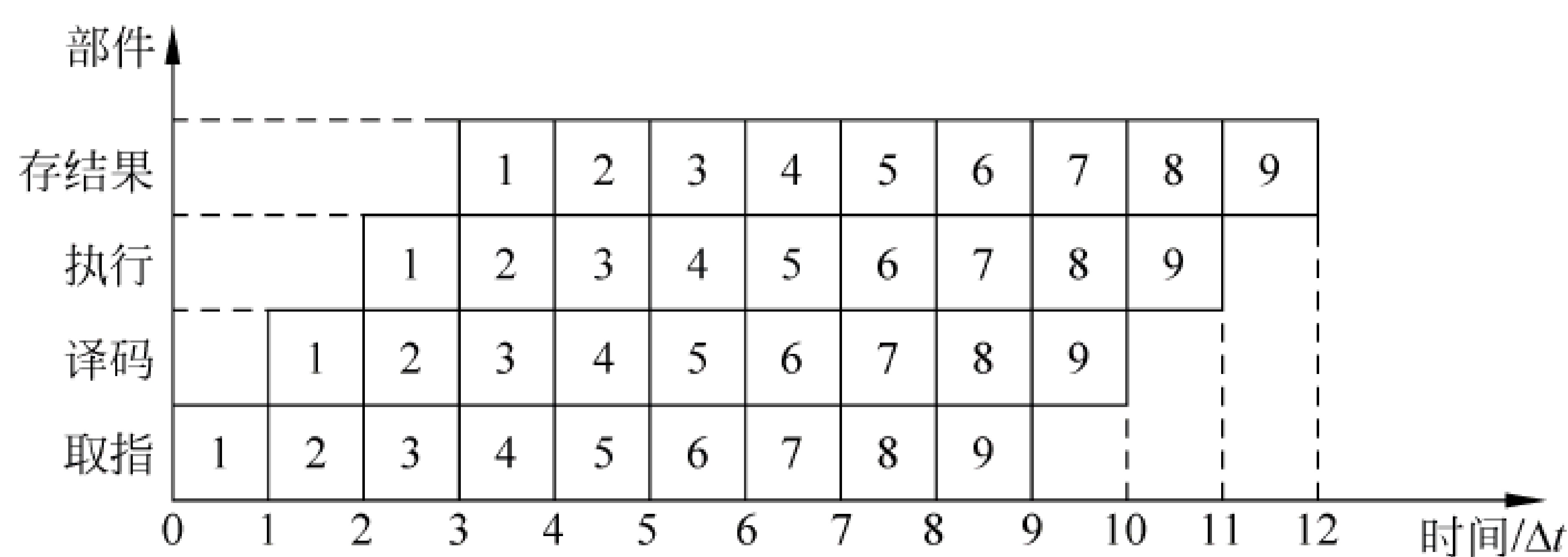


图 8-14　常规（度 $m=1$）标量流水处理机时空图

超标量处理机则采用多指令流水线，每个 Δt 同时流出 m 条指令（m 就是度）。图 8-15 是 $m=3$ 时的流水时空图，每 3 条指令为一组，执行完 9 条指令只需 $6\Delta t$。

超标量流水线处理机中配置多套功能部件、指令译码电路和多组总线，寄存器也备有多个端口和多组总线。程序运行时由指令译码部件检测顺序取出的指令之间是否存在数据相

关和功能部件争用情况，将可以并行的相邻指令送往流水线。度 $m=1$ 是超标量流水机的特例，并行度为 1 就逐条执行。超标量流水线处理机较容易实现，主要靠编译来优化编排指令的顺序，将可并行的指令搭配成组，硬件就不再调整指令的顺序。

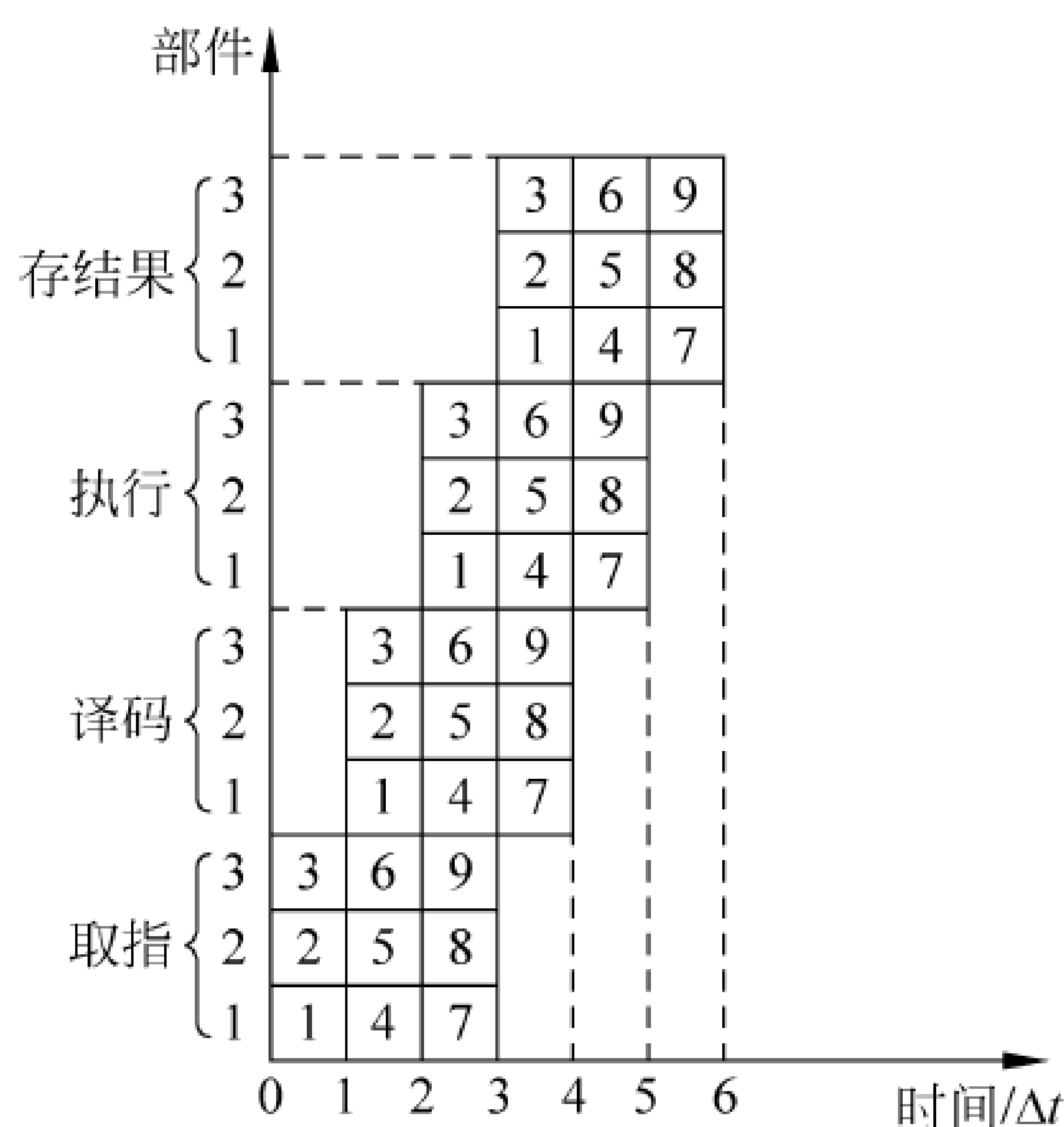


图 8-15　$m=3$ 时的超标量流水时空图

超标量流水线处理机的典型代表有 Intel 公司的 i860、i960、Pentium 处理机，Motolora 公司的 MC88110，IBM 公司的 Power 6000，Sun 公司的 SuperSPARC 等。

2. 超流水线处理机

如果超流水线处理机的度用 m 表示，一个机器周期为 Δt，那么把机器周期分为 m 个子周期，每个子周期表示为 $\Delta t'$，$\Delta t'=\Delta t/m$，那么每个 $\Delta t'$ 可以流出一条指令。用 k 表示一条指令所含的基本机器周期数，那么一条指令需花 $km\Delta t'$ 的时间，如图 8-16 所示。

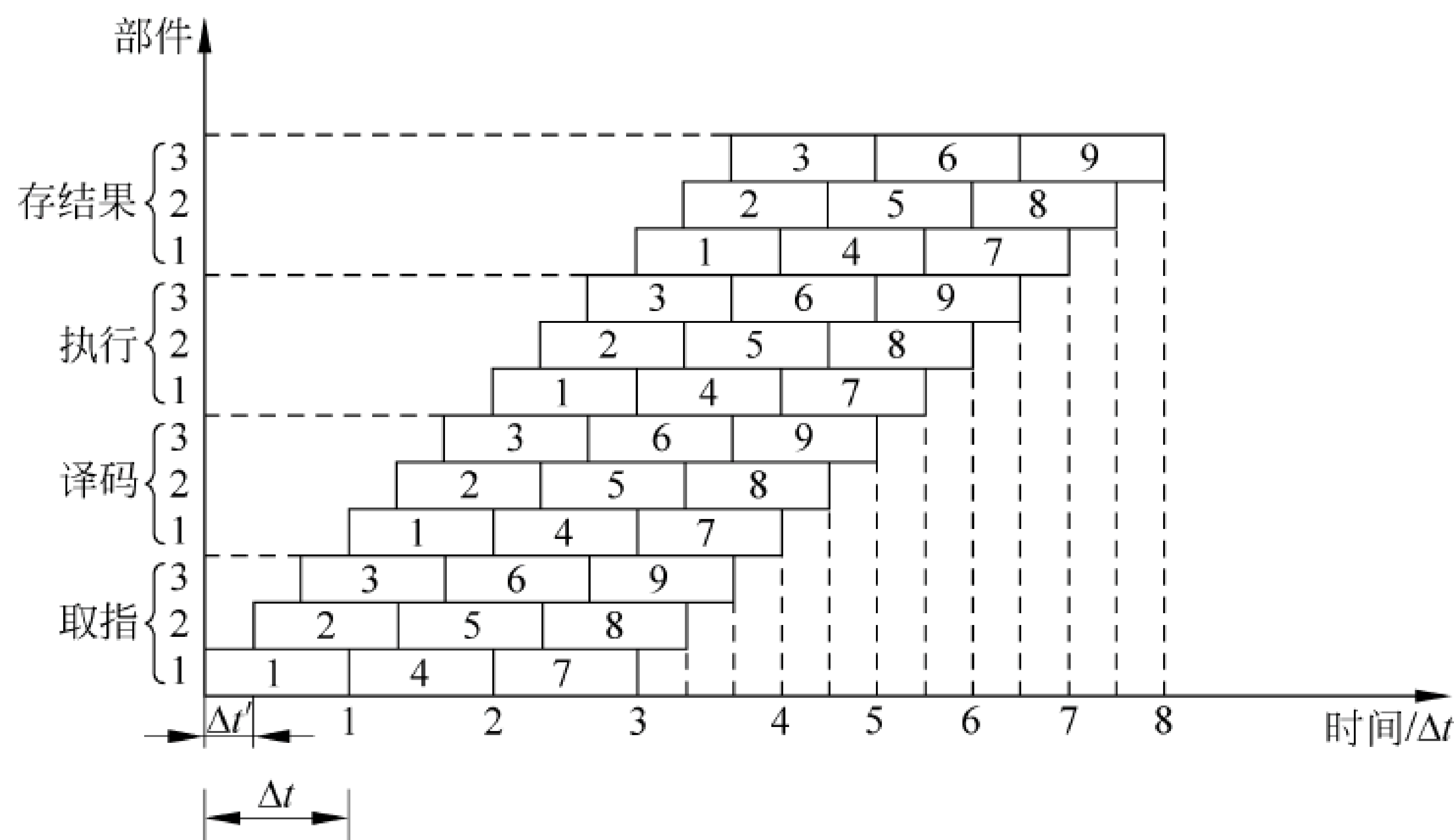


图 8-16　度 $m=3$ 的超流水线时空图

与前面的超标量处理机相比，超标量处理机着重利用资源重复，设置多个执行部件寄存器堆端口，而超流水线处理机则着重开发时间的并行性。

超流水线处理机的典型代表有 SGI 公司的 MIPS R4000、R5000、R10000 等。

3. 超标量超流水线处理机

将前面所讲的超标量流水线与超流水线机相结合就形成了超标量超流水线处理机。超标量超流水线处理机在一个 $\Delta t'$时间内发射了 k 条指令(超标量)。每次发射时间错开 $\Delta t'$(超流水),相当于每拍 Δt 流出了 nk 条任务,并行度为 $m=kn$。

例如,如图 8-17 所示,$k=3$,$n=3$,并行度 $m=9$,完成 12 个任务就需 $5\Delta t$,完成 21 个任务就只需 $6\Delta t$。任务越多,超标量超流水线处理机的性能优势就越明显。

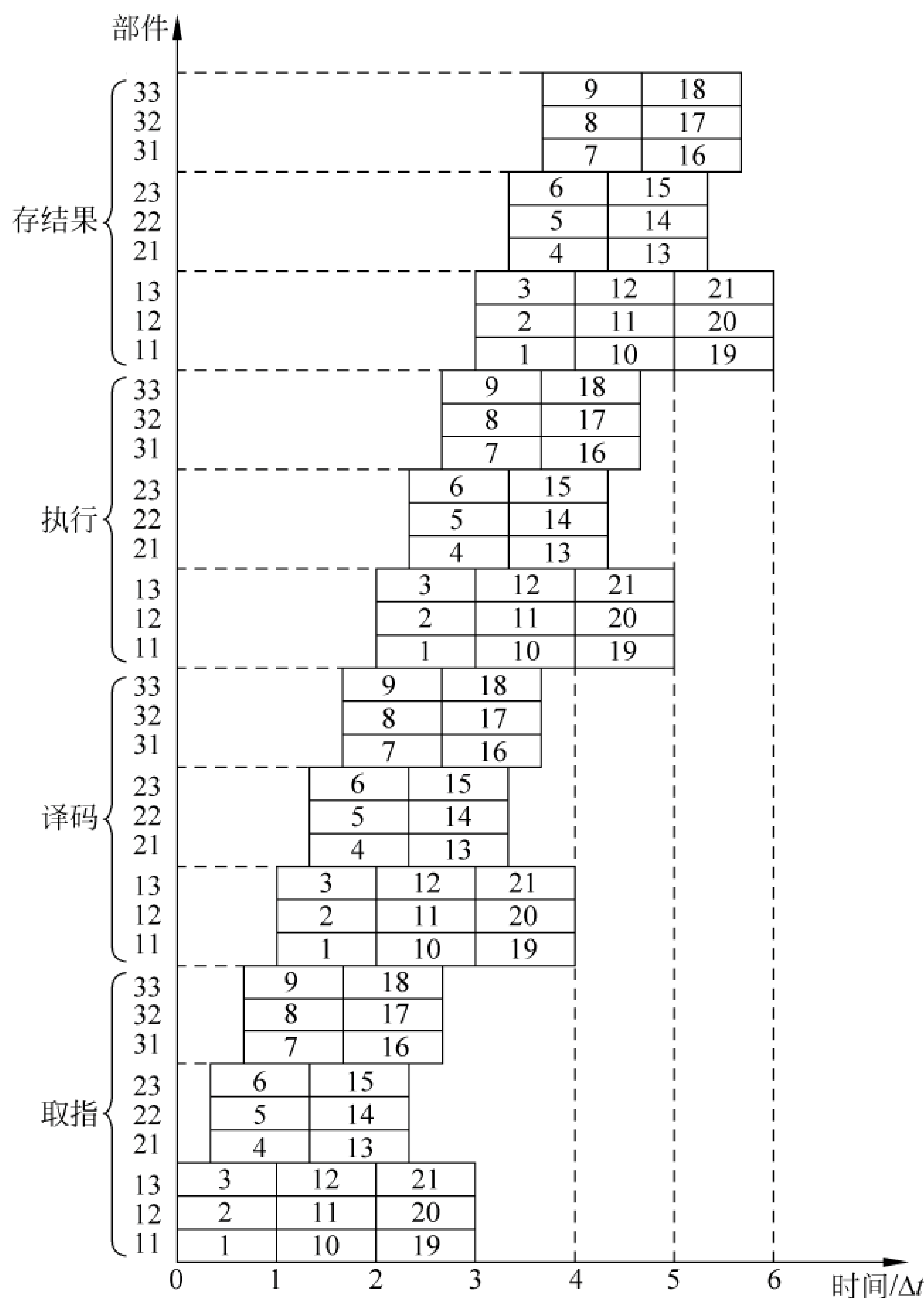


图 8-17　并行度 $m=9$ 的超标量超流水线时空图

超标量超流水线处理机的典型代表有 DEC 公司的 Alpha 等。

超标量是通过内置多条流水线来同时执行多个处理器的,其实质是以空间换取时间的。而超流水线是通过细化流水、提高主频,使得在一个机器周期内完成一个甚至多个操作,其实质是以时间换取空间的。如 Pentium 4 的流水线就长达 20 级。流水线设计的步(级)越长,其完成一条指令的速度就越快,因此才能适应工作主频更高的 CPU。但是流水线过长也会带来了一定的副作用,很可能会出现主频较高的 CPU 实际运算速度较低的现象,Intel

的 Pentium 4 就出现了这种情况，虽然它的主频可以高达 1.4GB 以上，但其运算性能却远远比不上 AMD 1.2GB 的速度甚至 Pentium Ⅲ。

4. 龙芯 2E 增强型处理器芯片的流水线

龙芯 2E(Longson 2E)是一款国产实现 64 位 MIPS Ⅲ指令集的通用 RISC 处理器，如图 8-18 所示。采用 90nm 的 CMOS 工艺，布线层为七层铜金属，芯片晶体管数目为 4700 万，芯片面积 6.8mm×5.2mm，最高工作频率为 1GHz，典型工作频率为 800MHz，实测功耗 5～7W。龙芯 2E 具有片上 128KB 一级缓存、512KB 二级缓存，单精度峰值浮点运算速度为 80 亿次/秒，双精度浮点运算速度为 40 亿次/秒，在 1GHz 主频下 SPEC CPU 2000 的实测分值达到 500 分，综合性能已经达到高端 Pentium Ⅲ以及中低端 Pentium 4 处理器的水平。

图 8-18　龙芯 2E 处理器

龙芯 2E 的基本流水线包括取指、预译码、译码、寄存器重命名、调度、发射、读寄存器、执行、提交 9 级，每一级流水包括以下操作。

(1) 取指流水级用程序计数器(PC)的值去访问指令 cache 和指令 TLB，如果指令 cache 和指令 TLB 都命中，则把 4 条新的指令取到指令寄存器(IR)。

(2) 预译码流水级主要对转移指令进行译码并预测跳转的方向。

(3) 译码流水级把 IR 中的 4 条指令转换成龙芯 2E 的内部指令格式送往寄存器重命名模块。

(4) 寄存器重命名流水级为逻辑目标寄存器分配一个新的物理寄存器，并将逻辑源寄存器映射到最近分配给该逻辑寄存器的物理寄存器。

(5) 调度流水级将重命名的指令分配到定点或浮点保留站中等待执行，同时送到 ROQ 中用于执行后的顺序提交；此外，转移指令和访存指令还分别被送往转移队列和访存队列。

(6) 发射流水级从定点或浮点保留站中为每个功能部件选出一条所有操作数都准备好的指令；在重命名时操作数没准备好的指令，通过侦听结果总线和 forward 总线等待它的操作数准备好。

(7) 读寄存器流水级为发射的指令从物理寄存器堆中读取相应的源操作数送到相应的功能部件。

(8) 执行流水级根据指令的类型执行指令并把计算结果写回寄存器堆；结果总线还送往保留站和寄存器重命名表，通知相应的寄存器值已经可以使用了。

(9) 提交流水级按照 Reorder 队列记录的程序的顺序提交已经执行完的指令，龙芯 2E 最多每拍可以提交 4 条指令，提交的指令送往寄存器重命名表用于确认它的目的寄存器的重命名关系并释放原来分配给同一逻辑寄存器的物理寄存器，并送往访存队列允许那些提交的存数指令写入 cache 或内存。

上述是其基本指令的流水级，对于一些较复杂的指令，如定点乘除法指令、浮点指令以及访存指令，在执行阶段需要多拍。

8.3　多处理机系统

多处理机系统属于并行处理技术中的一种空间并行,它通过使用多个处理机同时执行程序从而提高系统的性能。本节首先介绍多处理机系统的分类,然后介绍多处理机系统的相关技术,如由多处理机引起的 cache 一致性、多处理机操作系统及多处理机系统的并行性实现等。

8.3.1　多处理机系统分类

具有能同时执行多个任务或多条指令或同时对多个数据项进行处理的计算机系统统称为并行处理计算机系统,包括阵列计算机、向量计算机、多处理机系统和多计算机系统。

按照前面讲到的 Flynn 分类法,计算机系统可分成 4 类,即单指令流单数据流(SISD)系统、单指令流多数据流(SIMD)系统、多指令流单数据流(MISD)系统和多指令流多数据流(MIMD)系统。阵列计算机和向量计算机属于 SIMD 系统,它们通过使用多个处理器同时对多个数据进行处理,从而提高机器的数据处理能力,这类机器对于数组或向量运算具有较高的性能,常用于如图像处理、具有向量化运算的科学计算领域等。

多处理机系统和多计算机系统都属于 MIMD 系统,但相比较而言,两者具有很大的不同。多处理机系统是指两个或两个以上处理机通过高速互联网络连接起来,在统一的操作系统管理下,实现指令以上级(任务级、作业级)并行。多计算机系统则是由多个独立的计算机组成的,它们通过某种方式连接起来,实现并行处理或计算。一般来讲,多计算机系统属于松耦合系统,构成系统的可以是独立的计算机;而多处理机系统更多属于紧耦合系统,各处理机既独立又联系紧密,如通过共享存储器互连等。

从目前的实际应用来看,MIMD 系统是并行处理计算机系统中的主流,而多处理机系统又在 MIMD 系统中占据了主导地位。

MIMD 之所以能成为现代高性能计算机系统的主流体系结构,主要是因为以下两点:

(1) 灵活性好。通过适当的软硬件支持,MIMD 可以用作单用户机器,针对一个应用程序发挥其高性能;也可以用作多道程序机器,同时运行多个任务;还可以是这两种功能的组合。

(2) 性价比好。MIMD 可以充分利用商品化微处理器在性能价格比方面的优势。实际上,现有的多处理机系统几乎都采用与工作站和单处理机服务器相同的微处理器作为其处理器。

根据多处理机系统的组成结构来分,现有的多处理机系统主要包括:对称式共享存储器结构多处理机系统(Symmetric shared-memory MultiProcessor,SMP)、分布式共享存储器结构多处理机系统(Distributed Shared Memory,DSM)和大规模并行处理机(Massively Parallel Processor,MPP)等。

1. 对称式共享存储器多处理机

通常 SMP 系统使用商品微处理器(具有片上或外置高速缓存)作为其处理机,它们经由互联网络与一个共享存储器互连。共享存储器可以被所有处理器通过互联网络进行访

问，就如同一个单处理器访问它的存储器一样。所有处理器对任何存储单元有相同的访问时间。互联网络可以是单总线、多总线或者交叉开关。因为对共享存储器的访问是平衡的，每个处理器有相等的机会读/写存储器，也有相同的访问速度，故这类系统就称为对称多处理机(SMP)。因为这种对称多处理器的存储器是共享的，所以又称为共享存储器多处理机系统。SMP 系统的体系结构如图 8-19 所示。

对称多处理器的优点是并行度高。因为系统是对称的，每个处理器可等同地访问共享存储器、I/O 设备和操作系统服务。这种对称使得系统能开拓较高的并行度。但因为共享存储器，系统总线的带宽是有限的，就限制了系统中的处理器不能太多(一般少于 64 个)，同时总线和交叉开关互连一旦做成也难以扩展。

2. 分布式共享存储器多处理机

分布式共享存储器多处理机(DSM)具有分布的物理存储器。为支持更大数目的处理机，存储器必须分布到各个处理机上，而非集中式，否则存储器系统将不能满足处理机带宽的要求。系统将物理上分散的各台处理机所拥有的局部存储器在逻辑上统一编址，形成一个统一的虚拟地址空间，以实现存储器的共享。其存储器采用 cache 目录表来支持分布高速 cache 的一致性。系统中每个节点均包含了处理机、存储器、I/O 以及互联网络接口。DSM 系统的体系结构如图 8-20 所示。

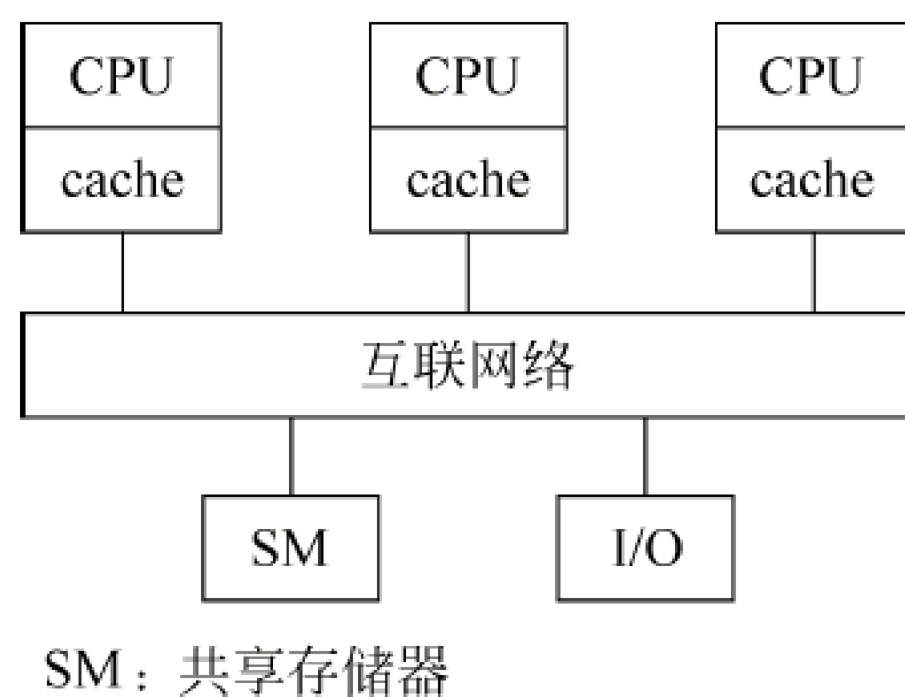


图 8-19 对称式共享存储器结构多处理机系统

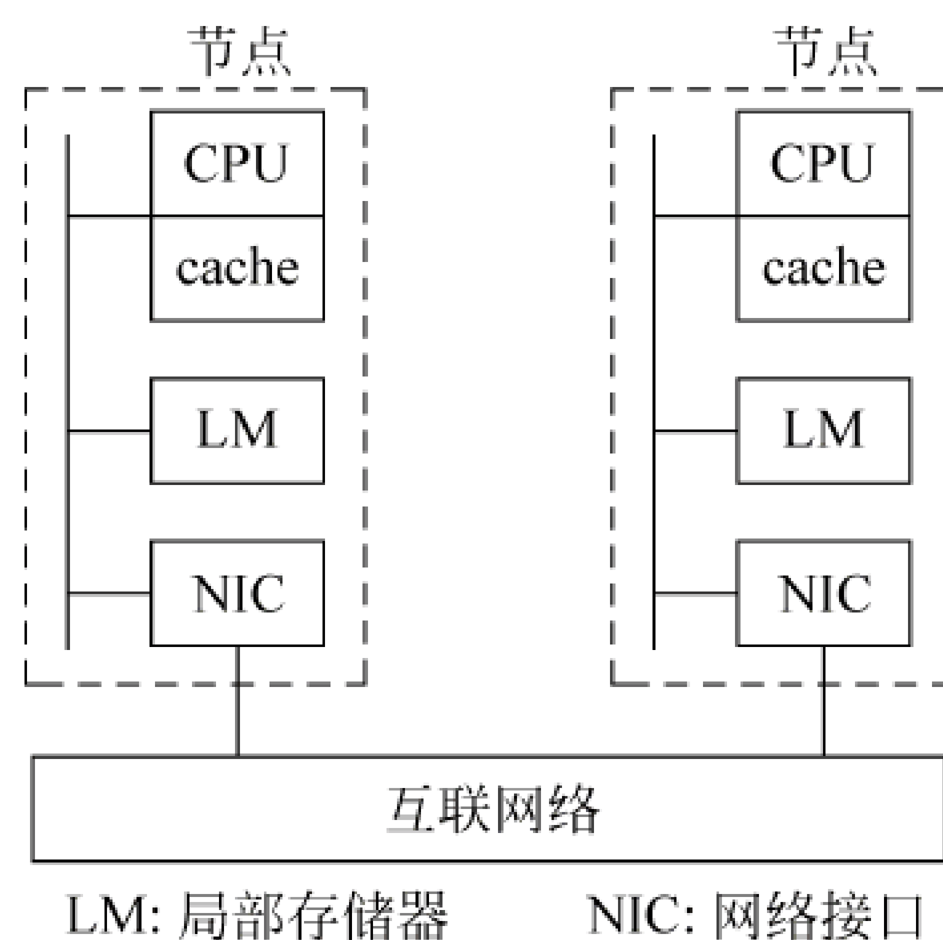


图 8-20 分布式共享存储器结构多处理机系统 DSM

将存储器分布到各节点有两个好处：第一，如果大多数的访问针对本节点的局部存储器，则可降低对存储器和互联网络的带宽要求；第二，对局部存储器的访问延迟低。

DSM 和 SMP 的主要差别是：SMP 中的处理机没有自己的局部存储器，系统的存储器是由所有处理机共享的；而 DSM 将存储器在物理上分布到各处理机中，并进行统一编址，形成一个共享的虚拟存储器。随着处理器性能的迅速提高和处理器对存储器带宽的要求不断提高，甚至在较小规模的多处理机系统中，采用分布式存储器结构也优于采用对称式共享存储器结构。

3. 大规模并行处理机

庞大的数据量、异常复杂的运算、极不规则的数据结构和极高的处理速度，这些是高科

技应用领域对计算机和通信网络在计算、处理和通信性能上不断提出的更高的要求；尤其是科学计算中的重大课题都要求计算机系统能提供 3T 性能，即 Teraflops 计算能力，TeraByte 主存储器，TeraByte/s 输入输出频带宽度；于是伴随着 VLSI 技术和微处理技术的发展，大规模并行处理机(MPP)就成了 20 世纪 80 年代中期计算机发展的热点。

MPP 一般是指超大型(Very Large-Scale)并行计算机系统，大规模并行处理需要有新的计算方法、存储技术、处理手段和结构组织方式。实现的方法是将数百乃至数千个高性能、低成本的 RISC 微处理器用专门的互联网络互连，组成大规模并行处理机(MPP)。这种处理机可进行中粒度和细粒度大规模并行处理，构成 SIMD 或 MIMD 系统，其优点在于它具有高性价比，并且可扩展性很好。

MPP 一般具有以下特性：

(1) 处理节点采用商品微处理器；

(2) 系统中有物理上的分布存储器；

(3) 采用高通信带宽和低延迟的互联网络(专门设计和定制的)；

(4) 能扩放至成百上千个处理器；

(5) 它是一种异步的 MIMD 机器，程序由多个进程组成，每个都有其私有地址空间，进程间采用传递消息相互作用。

大规模并行处理机(MPP)系统采用的技术主要是 VLSI、可扩展技术和共享虚拟存储技术；其适用的领域主要是科学计算、工程模拟和信号处理等以计算为主的一些重大课题和领域。比如全球气候预报、基因工程、飞行动力学、海洋环流、流体动力学、超导建模、半导体建模、量子染色动力学、视觉等。

8.3.2　多处理机的 cache 一致性

在单处理机系统中，cache 一致性问题只存在于 cache 与主存之间，即使有 I/O 通道共享 cache 亦可通过全写法或回写法较好地加以解决。在紧耦合多处理机系统中，如果采用全写法，也只能维持一个 cache 和主存之间的一致性，不能自动更新其他处理机中的 cache 的相同副本，所以解决不了多处理机中 cache 之间的一致性。解决多处理机系统中 cache 的一致性问题是研究多处理机技术中的一个重要问题。

1. 实现 cache 一致性的基本方案

实现 cache 一致性的方法有多种，大体分为两类：一类是软件方法，另一类是硬件方法。软件解决方法的思路是：在程序的编译阶段解决问题，即使用编译程序对程序代码进行 cache 一致性分析，确定什么样的数据项可能会造成 cache 的不一致性，然后相应地标记出这些项，最后由操作系统或硬件来防止这些数据项用于 cache。

软件方法的优点是：cache 的一致性问题的解决在编译时实现，使复杂的机器硬件系统设计变得简单。其缺点是：在编译时进行的保守判决会导致一些不会引起 cache 一致性的数据项可能也会禁止被 cache 引用，从而导致 cache 利用率的下降。一般来讲，硬件的方法比软件的方法效率更高，而且，硬件的方法对于程序员和编译程序来说都是透明的，也减轻了软件研制的开销。因此，在多处理机系统中更多采用的是硬件方法。

基于硬件的方法又称为 cache 一致性协议，通过对运行过程中共享数据块状态的跟踪

和潜在的不一致条件的动态识别，来阻止相关数据对 cache 的使用。硬件的方法又分成两大类：目录协议(Directory Protocol)和监听协议(Snoopy Protocol)。这些方法主要体现在具体实现上的不同，如数据块状态信息保存在何处，信息是如何组织的，在何处实施一致性以及实施一致性机构的组织等。

目录协议是在主存中保存一个目录，记录了共享数据块的状态及相关信息，由一个集中控制器(主存控制器的一部分)对该目录进行集中管理和维护，通过该目录来跟踪运行过程中共享数据块的状态，并对潜在的不一致条件进行动态识别。

监听协议是每个处理机 cache 除了包含主存储器中的数据拷贝之外，也保存着各个数据块的共享状态信息，各个处理机的 cache 控制器通过监听共享存储器总线来判断是否有总线上请求的数据块，从而对这些数据块进行跟踪和处理。

在使用多个处理机，且每个处理机的 cache 都与共享存储器相连组成的多处理机系统中，一般都采用以上基于写作废模式的监听协议。

2. 监听协议

可以使用两种方法来维持以上所讲的一致性要求：写作废协议和写更新协议。

写作废协议(Write Invalidate)是在一个处理机写某数据块后，若新写入主存储器中的数据与其他 cache 拷贝中的数据不一致，则使所有其他 cache 拷贝中的数据作废。这样的话，今后某个处理机对该数据进行读时会产生读失效，于是从主存储器中将该块调入 cache，该处理机的 cache 又与主存储器保持一致了。

写更新协议是当一个处理机写某数据项时，通过广播使其他处理机的 cache 中所对应的数据项拷贝也同时更新。为减少协议所需的带宽，应知道 cache 中该数据项是否为共享状态，也就是别的处理机 cache 中是否也存在该数据项拷贝。如果不是共享数据，则写时就无须进行广播。

在基于总线的多处理机中，总线和存储器带宽是最紧缺的资源，因此写作废协议成为绝大多数系统设计的选择。当然，在设计小数目处理机(2～4 个)的多处理机系统时，各个处理机紧密相连，更新所要求的带宽还可以接受。尽管如此，考虑到存储器性能和相关带宽要求的增长，更新模式很少采用。

对于监听协议，当某个处理机进行写数据时，必须先获得总线的控制权，然后将写入的数据块地址放到总线上，该地址也是其他处理机要作废的 cache 数据块地址。其他处理机一直监听总线，它们检测该地址所对应的数据是否在它们的 cache 中。若在，则作废相应的数据块，否则不予理睬。

当某个处理机写 cache 未命中时，除了将其他处理机上相应的 cache 数据块作废外，还要从主存储器取出该数据块。对于采用全写法的 cache，因为所有写 cache 的数据同时被写回主存，则从主存中总可以取到最新的数据值。而对于采用写回法的 cache，得到数据的最新值就会困难一些，因为最新值可能在某个处理机的 cache 中，也可能在主存中。在 cache 失效时可使用相同的监听机制，每个处理机都监听放在总线上的地址，如果发现某个处理机含有被请求数据块的一个已修改过的拷贝，它就将这个数据块送给发出读请求的处理机，并停止其对主存的访问请求。

进入 20 世纪 90 年代以后，几乎所有的通用微处理机芯片上都添加了支持监视 cache

协议的功能。采用最多的是一种基于监听协议的 MESI 协议。该协议是用协议中用到的 4 种状态即 Modified(修改)、Exclusive(独占)、Shared(共享)和 Invalid(无效)的首字母来命名的。这个协议中每个 cache 项都处于下面 4 种状态之一。

(1) 修改(Modified):该项的数据是有效的,但内存中的数据是无效的,而且在其他 cache 项中没有该项数据的拷贝。

(2) 独占(Exclusive):没有其他的 cache 项包括这行数据,内存中的数据是最新的。

(3) 共享(Shared):多个 cache 项中都有这行数据,内存中的数据才是最新的。

(4) 无效(Invalid):该 cache 项包含的数据无效。

大多数 x86 架构的 CPU 都采用这种 MESI 协议,如 Power PC601、M88110、Intel 的 Pentium 和 i860、AMD K6 及以后的一些产品中,都采用了此种方法。

8.3.3 多处理机操作系统

多处理机操作系统目前主要有三种类型:主从式操作系统、独立监督式操作系统和浮动监督式操作系统。

1. 主从式操作系统

主从式操作系统(Master-Slave)由一台主处理机记录、控制其他从处理机的状态,并分配任务给从处理机。例如,Cyber-170 采用的就是主从式多处理机操作系统,它驻留在一个外围处理机 P_0 上运行,其余所有的处理机包括中心处理机都从属于 P_0。另一个例子是 DEC System 10,它有两台处理机,一台为主,另一台为从。操作系统在主处理机上运行,从处理机的请求通过线路传送给主处理机,然后主处理机回答并执行相应的服务操作。主从式操作系统的监控程序及其提供服务的过程不必迁移,因为只有主处理机利用它们。当不可恢复错误发生时,系统很容易导致崩溃,此时必须重新启动主处理机。由于主处理机的责任重大,当它来不及处理进程请求时,其他从属处理机的利用率就会随之降低。

主从式操作系统有以下特点:

(1) 操作系统程序在一台处理机上运行。如果从处理机需要主处理机提供服务,则向主处理机发出请求,主处理机接受请求并提供服务。不一定要求把整个管理程序都编写成可重入的程序代码,因为只有一个处理机在使用它,但有些公用例程必须是可重入的才行。

(2) 不存在管理表格存取冲突和访问阻塞问题。

(3) 当主处理机故障时很容易引起整个系统的崩溃。如果主处理机不是固定设计的,管理员可从其他处理机中选一个作为新主处理机并重新启动系统。

(4) 任务分配不当容易使部分从处理机闲置,从而导致系统效率下降。

(5) 系统由一个主处理机加上若干从处理机组成,硬件和软件结构相对简单,但灵活性差。

(6) 主从式操作系统用于工作负载不是太重或由功能相差很大的处理机组成的非对称系统。

2. 独立监督式操作系统

独立监督式操作系统(Separate Supervisor)与主从式操作系统不同,在这种类型中,每

一个处理机均有各自的管理程序(核心)。

独立监督式操作系统有以下特点:

(1) 每个处理机将按自身的需要及分配给它的任务的需要来执行各种管理功能,这就是所谓的独立性。

(2) 由于有好几个处理机在执行管理程序,因此管理程序的代码必须是可重入的,或者为每个处理机装入专用的管理程序副本。

(3) 因为每个处理机都有其专用的管理程序,故访问公用表格的冲突较少,阻塞情况自然也就较少,系统的效率就高。但冲突仲裁机构仍然是需要的。

(4) 每个处理相对独立,因此一台处理机出现故障不会引起整个系统的崩溃。但是,要想补救故障造成的损失或重新执行故障机未完成的工作则非常困难。

(5) 每个处理机都有专用的 I/O 设备和文件等。

(6) 独立监督式操作系统要实现处理机负载平衡更加困难。这类操作系统适合于松耦合多处理机体系,因为每个处理机均有一个局部存储器用来存放管理程序副本,存储冗余太多,利用率不高。

3. 浮动监督式操作系统

浮动监督式操作系统(Floating Supervisor)每次只有一台处理机作为执行全面管理功能的“主处理机”,但根据需要,主处理机是可浮动的,即从一台切换到另一台处理机。这是最复杂、最有效、最灵活的一种多处理机操作系统,适用于紧耦合多处理机体系,常用于对称多处理机系统中。

浮动监督式操作系统有以下特点:

(1) 每次只有一台处理机作为执行全面管理功能的“主处理机”,但容许数台处理机同时执行同一个管理服务子程序。因此,多数管理程序代码必须是可重入的。

(2) 根据需要,“主处理机”是可浮动的,即从一台切换到另一台处理机。这样,即使执行管理功能的主处理机故障,系统也能照样运行下去。

(3) 一些非专门的操作(如 I/O 中断)可送给那些在特定时段内最不忙的处理机去执行,使系统的负载达到较好的平衡。

(4) 服务请求冲突可通过优先权办法解决,对共享资源的访问冲突用互斥的方法解决。

(5) 系统内的处理机采用处理机集合概念进行管理,其中每一台处理机都可用于控制任一台 I/O 设备和访问任一存储块。这种管理方式对处理机是透明的,并且有很高的可靠性和相当大的灵活性。

8.3.4 多处理机的并行性实现

多处理机主要实现作业之间、程序段之间、任务之间的并行,也可包含指令级、指令内部各微操作之间的并行。

多处理机的并行性可利用并行算法、并行程序设计语言、并行编译、并行操作系统以及指令硬件等多种途径来开发和实现。

1. 并行算法、并行语言及并行编译

1）并行算法

提高并行处理效率的关键之一是并行算法。算法需适应计算机的结构。如果一种算法所表达出来的并行度与计算机的并行度基本一致，便能提高计算机的解题效率。

一般算法规定了求解某一特定问题时的有穷运算处理步骤。并行算法则是指可同时执行的多个进程的集合，各进程可相互作用、协调和并发操作。

并行算法的分类标准不同则结果也不相同。

按运算的基本对象可分为数值型（基于代数运算）和非数值型（基于关系运算）两种。比如，矩阵运算、线性方程组求解等均为典型的数值型并行算法；非数值型并行算法主要是对符号操作的，以排序、选择、查找、字符处理的并行为主要代表。

按并行进程间的操作顺序的不同又可分为同步型、异步型和独立型三种。同步型并行算法意味着并行的各个进程间由于相关而必须依序等待；异步型并行算法则是各个进程间相互独立，没有关联，无须因关联而等待，进程的中止或继续执行取决于执行情况；独立型并行算法则意味着各个进程间完全独立，进程间无须通信。

按计算任务的大小还可以分为细粒度、中粒度和粗粒度三种。细粒度并行算法的典型代表是向量或循环级的并行；较大的循环级并行一般就属于中粒度并行算法了；粗粒度并行算法一般指子任务级的并行。

研究并行算法的一种思路是将大的程序分解成一定数量的可并行处理的子过程（可以是进程、任务或程序段），其直观的表现形式是把每个过程看成一个节点，将过程之间的关联用节点组成的树来描述。这样，程序内各过程之间的关系就可以简单地当成一种算术表达式中各项之间的运算关系来处理了，表达式中的每一项都可以看成一个程序段的运行结果。这样，程序段之间的并行性问题就可以被设想成是对算术表达式如何进行并行运算的问题了。

为了评价一个并行算法性能到底如何，需要定义一些标准。用 P 表示可以并行处理的处理机的数量；用 T_1 表示单台处理机顺序运算的级数，用 T_P 表示 P 台处理机运算的级数（树的高度）；则 S_P（此处指多处理机的加速比）就表示 T_1 与 T_P 之比；用 E_P 表示 P 台处理机的设备利用率（使用效率），$E_P=S_P/P$。

下面我们用一个例子来说明不同的并行算法对计算机性能的影响。

【例 8.1】 有一个算术表达式 $F_1=a+b\cdot x+c\cdot x^2+d\cdot x^3$，试分析算法对树高的影响。

解：单处理机上的执行一般会利用霍纳（Horner）法进行变换，得到等价式为 $F_1=a+x(b+x(c+x(d)))$，这是一个典型的循环算法，需 3 个乘加循环，6 级运算，即 $P=1$，$T_1=6$，适合于单处理机，用树状流程图表示，结果如图 8-21(a)所示。

但是，这样的变换并不适合在多处理机上并行运行，因为这 3 个乘加循环之间产生了关联，无法并行。还不如用原来的式子在多处理机上直接求解更有效，此时 $P=3$，$T_P=4$，$S_P=3/2$，$E_P=1/2$。如图 8-21(b)所示，两者比较，加速比 $S_P=3/2$，但 $E_P=1/2$，这就是说运算的加速总是伴随着效率的下降。

既然把运算过程表示成了树，树的高度又代表着运算的级数，那么提高运算的并行度问

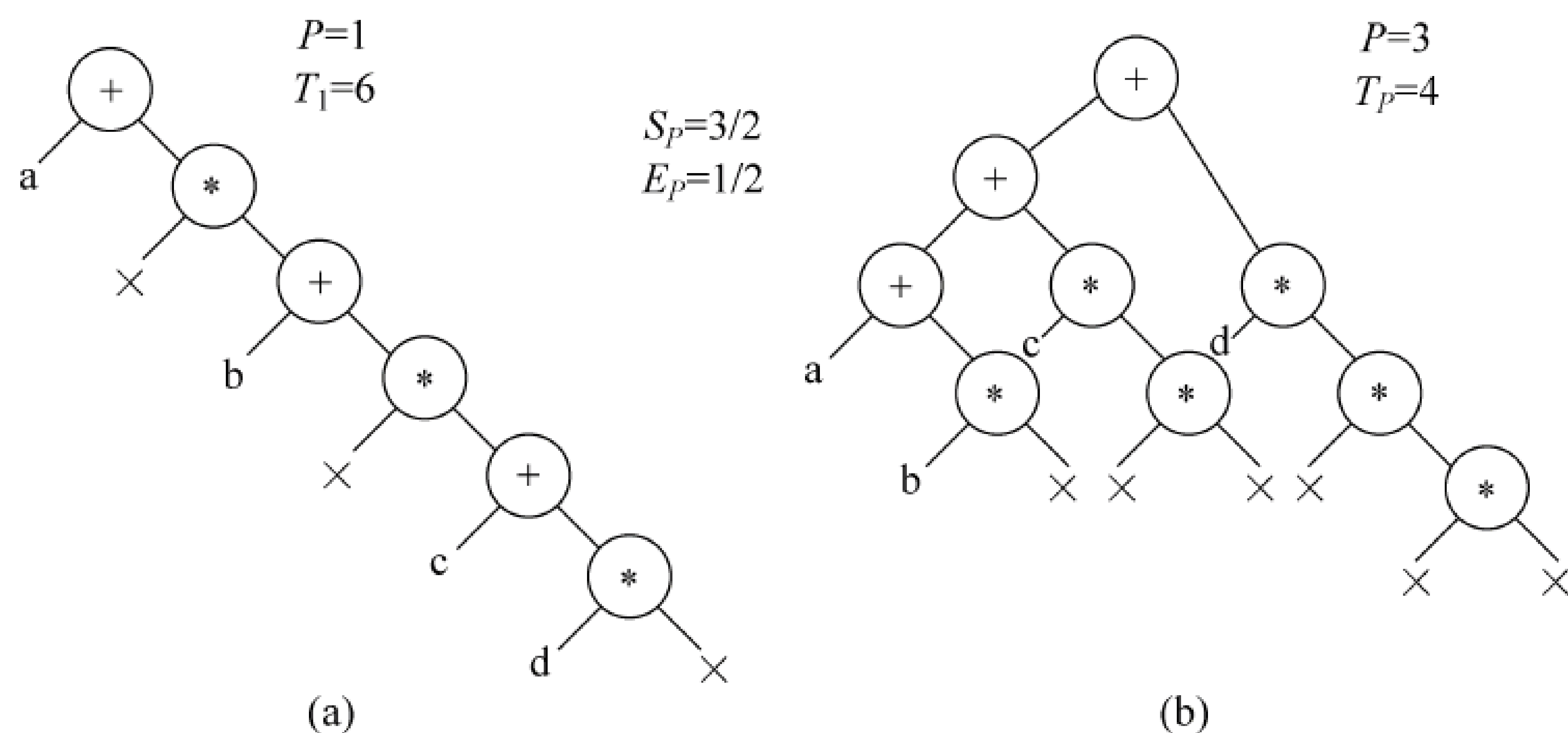


图 8-21 算法对树高产生影响的树状流程图

题就变成了如何对树进行变换,来降低树的高度,以减少运算级数的问题了。那么如何对树进行变换呢?

树状结构可以用结合律、交换律和分配律来变换树的形状。对树状结构进行变换的过程是:首先利用交换律把相同的运算集中在一起。然后利用结合律把参加这些运算的操作数配对,尽可能并行运算,从而组成树高最小的子树。最后再把这些子树结合起来,用分配律进一步降低树高。

采用多处理机的目的就是要提高速度,这同时要求尽可能把任务分化成更多可并行执行的过程。上面已经把任务并行化设想成是运算并行性问题,并用树表示了。那么对树而言,就是要尽可能增大树每一层的节点数,即增大树的广度,这样就可以使各处理机可并行的过程数尽可能增大,树的高度就降低了,也就是处理机运算的级数也降低了。一句话,就是当最大限度地降低了树的高度之后,在这个基础上再缩小树的广度,目的就是在达到一定的加速比之后,再通过减少处理机数量 P 降低来多处理机的效率。

【例 8.2】 有一个表达式为 $F_2=m+b\cdot c+b\cdot d\cdot e\cdot f+b\cdot g+n$,分析求解其并行算法。

解:通常可以从算术表达式本身出发,利用交换律把相同的运算集中在一起,然后再利用结合律把参加运算的操作数配对,尽可能并行运算,从而组成树高最小的子树,再把这些子树结合起来即可。

本题,$F_2=m+b\cdot c+b\cdot d\cdot e\cdot f+b\cdot g+n=m+b(c+d\cdot e\cdot f+g)+n$,用单处理机需 7 级运算;利用交换律和结合律可改写成 $F_2=(m+n)+b((c+g)+d\cdot e\cdot f)$,此时 $P=2$,只需 5 级运算即可。且 $S_P=7/5$,$E_P=0.7$。

进一步分析发现用分配律还可以降低树高,只要恰当地平衡好各子树的级数就常常能收到很好的效果。

于是 F_2 表达式进一步被转换成 $F_2=(m+n)+(b\cdot c+b\cdot g)+b\cdot d\cdot e\cdot f$,从而,$P=3$,$T_P=4$,$S_P=7/4$,$E_P=7/12$。需 5 级运算,$S_P$ 提高了,但效率下降了。

2)并行语言和并行编译

并行算法需要用并行程序来实现。如何在程序中识别和表示并发进程部分呢?这就需要在程序中加入能明确表示并发进程的语义表示方法,而这就是下面要讨论的并行语言。

表达式运算并行性的识别和实现,除依靠并行算法还可以依靠编译程序,可以经过或不

经过逆波兰后缀表达式产生并行指令。

例如，算术表达式 $Z=M+A*B*C/D+N$，利用串行编译算法，产生三元指令组为

```
1  *  A  B
2  *  1  C
3  /  2  D
4  +  3  M
5  +  4  N
6  =  5  Z
```

指令之间都是相关的，需 5 级运算。但如果利用并行编译算法，其三元指令组为

```
1  *  A  B
2  /  C  D
3  *  1  2
4  +  M  N
5  +  3  4
6  =  5  Z
```

两个处理机并行处理的话，则三级运算就够了。

根据这个并行编译的结果，可以很容易地得到以下程序：

```
S1   G = A * B
S2   H = C/D
S3   I = G + H
S4   J = M + N
S5   Z = I + J
```

如果不加并行控制语句，这个程序仍然只是一个普通的串行程序。现在就来给这个程序加入块控制结构式语言，该语言由 E. W. Dijkstra 提出，在该语言中，把所有可并行执行的语句或进程用 Cobegin-Coend 或 parbegin-parend 括住，以示区别于其他串行执行的语句或进程。

Cobegin-Coend 或 parbegin-parend 成对使用，主要用于描述多程序多数据的并行，并行的各进程同时开始，但它们相互独立，不相关，它们也并不一定同时结束，仅当所有并发进程完成时才终止并行。

其一般形式为

```
Begin
S0;
Cobegin    S1; S2; …; Sn; Coend
Sn+1;
end
```

或

```
Begin
S0;
parbegin    S1; S2; …; Sn; parend
Sn+1;
end
```

于是，算术表达式 $Z=M+A*B*C/D+N$ 对应的并行控制程序如下：

```
Begin
Cobegin        S1; S2; S4;    Coend
S3;
S5;
end
```

2. 程序的并行性

在多处理机系统中，提高程序并行性的关键，是把任务分解成足够多的可同时操作的进程。在程序语言中，还须扩充能明确表达进程并发性的语句，以便程序运行时能为相应的控制机构提供控制和管理手段，其中包括并行任务的派生、通信和调度。ADA 语言为描述多处理机并行程序结构提供了必要的语句。为适应数据流计算机而出现的若干数据流语言如 Id 语言和 VAL 语言也已经在试用，其重要特点是把数组看成值而不是目标。用数据流语言编写的程序能够自然地表达出最大的运算并行性。

8.4 机群系统

机群系统是近些年来出现的一种并行处理技术，它利用高速网络将一些独立的机器系统以松耦合方式连接起来，构成一个大小规模可随应用的需要而变化的集群系统。本节首先介绍机群的概念，然后介绍机群系统的组成及相关技术，最后举一个机群的应用实例。

8.4.1 机群系统的定义

随着微处理器性能的不断提高和高速网络技术的发展，传统的松散耦合系统的通信瓶颈逐步得到了缓解；同时并行编程环境的开发使得新编并行程序或改写串行程序变得更为容易。这一切都导致一种新的并行处理系统——机群系统得以发展起来，并成为当前并行处理技术新的发展方向。

所谓机群系统(Cluster)是指一组完整的计算机互连，它们作为一个统一的计算机资源一起工作，并能产生一台机器的印象。术语"完整计算机"，意指一台计算机离开机群系统仍能运行自己的任务，机群系统中每台计算机一般称为节点。

根据定义，机群是一组独立的计算机节点的集合体，但机群对用户和应用来说只是一个单一的系统，节点间通过高性能的互联网络连接。各节点除了可以作为一个单一的计算资源供交互式用户使用外，还可以协同工作并表现为一个单一的、集中的计算资源供并行计算任务使用。

一般来讲，机群系统中主要包括下列组件：

(1) 高性能的计算节点机。机群的节点可以是一个工作站，或者是一台个人计算机，但也可以是一台规模相当大的对称多处理机(SMP)。

(2) 高速交换网络。节点间的互连可以通过普通的商用网络(如以太网、FDDI、ATM等)和使用标准的网络通信协议，也可以使用专门设计的网络。

(3) 具有较强网络功能的操作系统。

(4) 机群中间件。支持分布共享存储器及为用户提供单一系统映像的资源管理和调度软件等。

(5) 并行程序设计环境与工具,如编译器、语言环境、并行虚拟机(PVM)和消息传递接口(MPI)等。

(6) 应用,包括串行和并行应用程序。

机群系统提出之后发展得十分迅猛,已成为目前研究的热点。机群受到广泛关注的原因是多方面的,其中之一就是它可以用商品处理器和商品网络方便地构造。另外它还有许多过去的并行系统不可比拟的优势,主要体现在以下几个方面:

(1) 投资风险小。传统的大规模并行处理机比较昂贵,如果性能不好就等于浪费了大量资金。而机群即使作为并行系统效果不好,它的每个节点仍可以作为高性能微机使用,不会浪费资金。

(2) 性能价格比高。传统的并行机由于生产批量小往往价格昂贵。机群的节点和互联网络等可以使用商品化的计算机产品,无须专门单独设计和制造,这可以大大降低成本,因而机群的性能价格比好。在相同的性能峰值情况下,机群的价格比传统的多处理机系统可以低 1 到 2 个数量级。

(3) 可用性好。一方面系统的开发周期短,机群的硬件都是商用的,开发的重点在通信机制和并行编程环境上;另一方面编程方便,软件继承性好。在机群系统中用户无须学习新的并行程序设计语言。只需在常规的 C、C++、FORTRAN 串行程序中插入少量通信原语,即可使其在机群上运行。

(4) 可靠性高。机群中有多个存储器、处理机和磁盘部件,有一个部件出现故障,其他部件仍能使用,从而保证机群能继续工作。机群中每个节点都有自己的操作系统,一个操作系统崩溃了,其他节点仍能工作。这一点显然比 SMP 好,因为 SMP 的存储器如有故障,将会使整个系统不能工作。SMP 只有一套操作系统驻留在共享存储器中,操作系统崩溃,整个系统也不能工作。

(5) 可扩展性好。机群的计算能力能随节点的增加而增加。机群中的处理机、存储器、磁盘,甚至 I/O 设备都可增减。由于节点间是松耦合的,机群的节点数有可能增加到几百个,而 SMP 只能增减处理机,处理机数最多也只能有几十个,因为 SMP 是紧耦合的,存储器会成为系统的瓶颈。

(6) 系统结构灵活。不同结构、不同性能、不同操作系统的工作站都可以连接起来构成机群系统。这样用户就可以充分利用现有的设备以及闲散的计算机,用少量投资获得较大的计算能力。

但机群也有不足之处。由于机群是由多台完整的计算机组成的,因而它的维护工作量和费用较高,它相当于要同时去管理多个计算机系统。SMP 则显得比机群好,因为对于 SMP,管理员要维护的只是一个计算机系统。正因为如此,现在很多机群采用 SMP 作为节点,这样可以减少节点数,也就减少了维护工作量和开支。

8.4.2 机群系统的组成

如图 8-22 所示是一个典型的机群系统的组成结构图。首先,在硬件的组成上,各个节点(PC 或工作站)通过高速网络或硬件开关互连;其次,每个独立的节点计算机都有自己的

操作系统用于完成网络通信，并安装有中间件(Middleware)以允许机群操作；最后，通过机群系统提供的并行程序设计环境为并行应用程序提供运行环境。

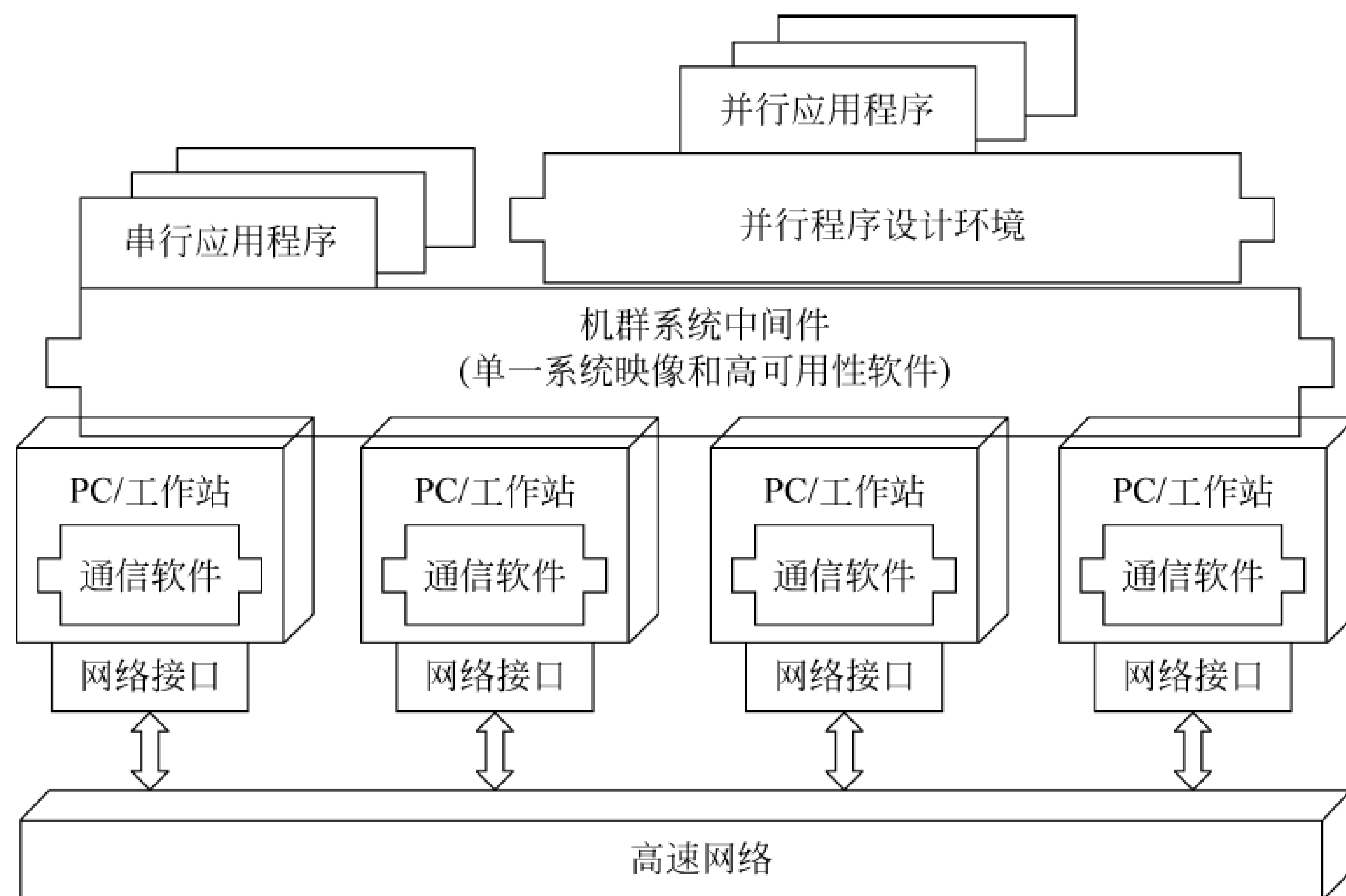


图 8-22 机群系统组成结构图

机群中的中间件能提供以下的功能。

(1) 单一入口点：用户通过单一入口登录到机群上而不是个别的计算机上。

(2) 单一文件层次：用户看到的是同一根目录下的单一文件目录系统。

(3) 单一控制点：系统使用一台默认的机器用于整个机群的管理和控制。

(4) 单一存储空间：机群为用户程序提供的是统一的共享存储器，虽然共享存储器可以分布在各个节点中，但对用户程序而言，它们所看到的是一个统一的虚拟存储空间。

(5) 单一作业管理系统：在机群作业调度程序管理下，用户提交作业无须指定执行此作业的宿主计算机。

(6) 单一用户接口：机群使用一个统一的公共图形接口支持所有用户，不管用户从哪台机器登录系统。

(7) 单一 I/O 空间：任一节点都能访问系统中所有的 I/O 设备或磁盘系统，而无须知晓其在系统中的具体位置。

(8) 进程迁移：这一功能可以使系统负载均衡。

8.4.3 机群系统中的关键技术

机群系统中的关键技术主要包括通信技术、并行程序设计环境、单一系统映像(Single System Image，SSI)等。

随着商品处理器性能的不断提高，机群各节点的处理速度已相当高，制约机群性能的主要因素是节点间的通信速度。如果通信速度跟不上，各节点的处理能力就发挥不出来。提高通信速度目前从硬件、软件方面都采取了措施。

硬件方面是尽量使用高速网络，如快速以太网、ATM、Myrinet(Myricom 公司从 1994 年就

开始销售其第一代 Myrinet 产品，当时只是作为以太网络的第二选择来实现机群系统中的节点互连，除了 100MB/s 的高带宽外，它的主要优势是小数据量延时，只有 10～15ms，这与当时 Convex、IBM、SGI 等公司在并行系统中使用的专有网络的高延迟形成鲜明对比。此后随着软硬件的不断升级，Myrinet 更是成为了机群制造商的第一选择）等。它们的带宽已达 Gb/s 数量级。

在软件方面，提高网络传输速度的主要措施是努力减小通信在软件方面的开销，包括精简通信协议，设计新的通信机制等。传统的 TCP/IP 协议是面向低速率、高差错和大数据包传输而设计的，其设计目标与机群系统的现实情况并不相符。TCP/IP 协议有很多层次，数据传输时需要反复拷贝，带来了很大的时延。另一方面，各层中有许多重复的操作，在机群环境中这些操作是不必要的。所以，通过修改、精简协议可以降低通信开销。例如，为了提高传输速度，UC Berkeley 开发了一种名为 Active Message 的通信机制。这种通信机制的思想是消息本身带有消息处理程序的地址和参数。消息到达目的节点后系统立即产生中断调用，并由中断处理机制启动消息处理程序。消息处理程序从网卡上取出消息并给发送方发送一个应答消息，然后返回原来被中断的程序。Active Message 是一种消息驱动的异步通信方式，能更好地实现通信与计算的重叠。

为了便于用户使用，机群系统必须给用户提供一个方便易用的并行程序设计环境。目前机群系统上广泛使用的并行程序设计环境有 PVM、MPI、Express（1982 年，Caltech 的一个 Hypercube 研究小组着手开发世界上第一台实用并行机系统 CosmicCube；稍后，这个小组开始研制相应的软件开发工具 Crystalline OS——CrOS，此即为 Express 的原型）、P4 等。它们都是基于消息传递方式的。其中 PVM 和 MPI 应用得最多。PVM（Parallel Virtual Machine）是美国橡树岭国家实验室及美国几所大学联合开发的并行计算工具软件，支持 C、C++ 和 FORTRAN。MPI（Message Passing Interface）是一个消息传递标准，是由 MPI 委员会在 1992 年 11 月至 1994 年 1 月举行的一系列会议上逐渐产生的，它的设计非常审慎，博采众长。PVM 和 MPI 都是免费软件，它们都可以方便地进行再开发。

单一系统映像（SSI）的目的是将整个机群系统虚拟为一个统一的系统，使用户感觉不到各工作站的存在，而好像就在使用一台普通的计算机。SSI 的作用是使分散的资源看起来是一个统一的更强大的资源。在机群系统中，SSI 可以用硬件实现也可以用软件实现。用软件实现时，是通过中间件来实现的。中间件是介于操作系统和用户层之间的一层软件。中间件与操作系统联系在一起，支持 SSI、通信、并行度、负载平衡等，在所有的节点上提供对系统资源的统一访问。

机群系统中还有一些其他的重要技术，例如，全局资源的利用。数据表明，由于网络速度的提高，节点访问其他节点的内存要比访问本地硬盘快。因此有效地利用整个系统的内存减少使用磁盘可以提高计算速度。这提出了一个如何有效地利用全局资源的问题。又如负载平衡问题，尤其在异构机群中，评价各节点的计算能力以及进程迁移都很困难。

需求推动技术的发展，从工业标准机群渐成主流这一事实，机群就已经向人们展示了它强大的生命力。

8.4.4 机群系统举例

IBM SP2 是用机群方法构成的专用机群系统，是机群中有代表性的产品，它既可进行

科学运算，也可供商业应用。1997年战胜世界国际象棋冠军卡斯帕洛夫的“深蓝”，就是一个采用30个IBM RS/6000工作站节点（带有专门设计的480片国际象棋芯片）的IBM SP2机群。

SP2机群采用的是一种分布式存储器MIMD体系结构，如图8-23所示。

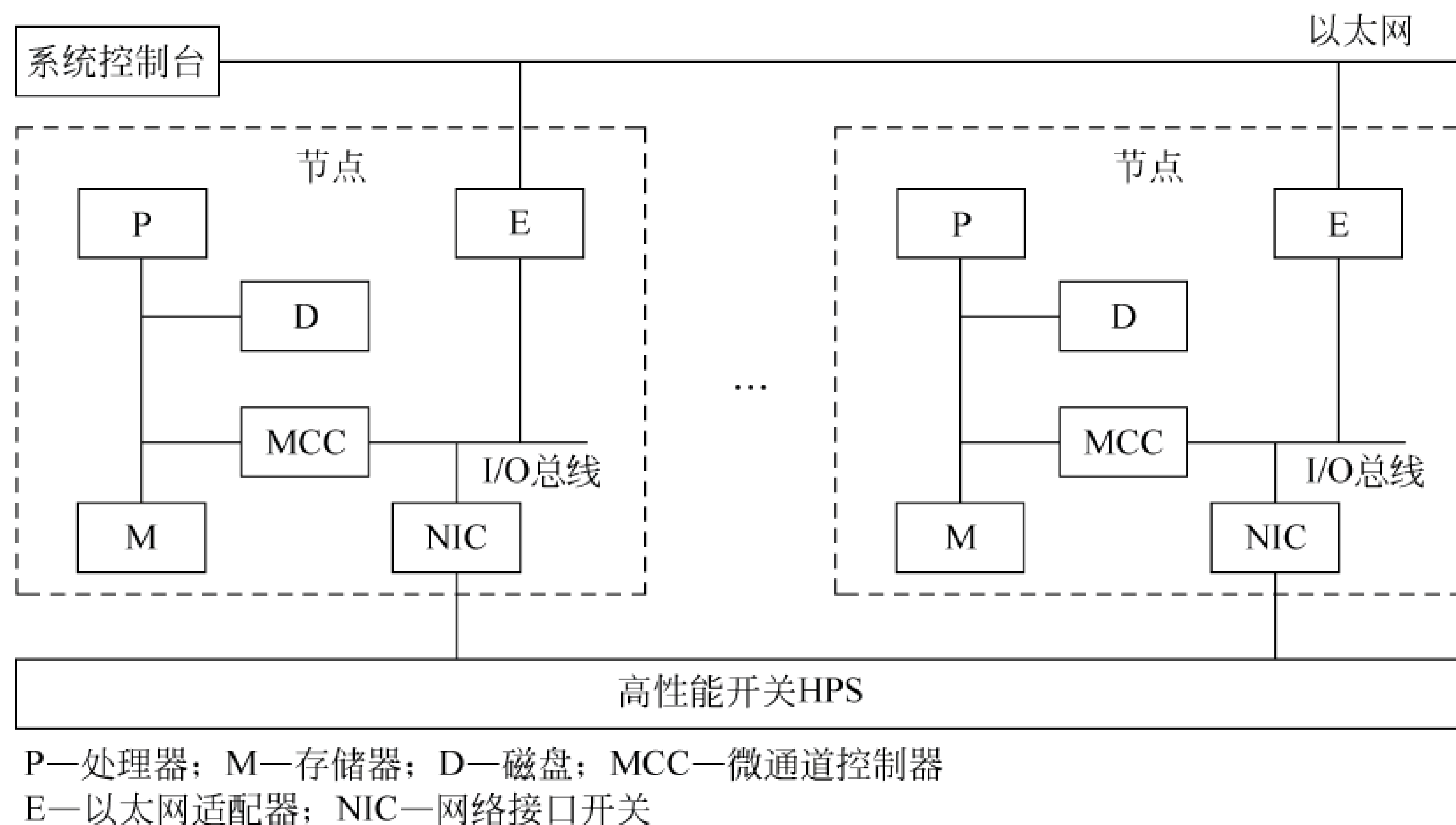


图 8-23 IBM SP2 结构

其中，SP2的每个节点是一台RS/6000工作站，带有自己的存储器和本地磁盘。每个节点配有一套完整的AIX操作系统（IBM的类UNIX操作系统），节点间的互联网络接口是松耦合的，通过节点本身的I/O微通道连接到网络上。节点的硬件和软件都能按不同用户的应用和环境的需要个别地进行配置。

SP2的节点数可以为2～512个。除了每个节点采用RS/6000工作站外，整个SP2系统还另外配置了一台RS/6000工作站作为系统控制台之用。节点间使用了两个网络进行互联，一个是标准的以太网，另一个是专门设计的高性能开关HPS（一种Omega多级开关网络）。以太网可以用于对通信速度要求不高的程序开发工作，开发好的程序在正式运行时用HPS。以太网还有备份的作用，当HPS有故障时可以通过以太网使系统维持工作。以太网还可以供系统的监视、引导、加载、测试和其他系统管理用。

从图8-23中可以看到每个节点有独用的存储器和本地的磁盘，各个节点通过微通道控制器（MCC）的I/O总线与外部网络连接，其中通过以太网适配器（E）连接到以太网上，通过网络接口开关（NIC）连接到HPS。NIC又称为开关适配器，它有一个8MB的DRAM用来存放各种不同协议所需的大量报文，并用一台i860微处理机进行控制。在SP2系统中，除HPS外，有的还采用光纤分布式数据接口（FDDI）环连接各节点。微通道是IBM公司的标准I/O总线，用于把外部设备连接到RS/6000工作站和IBM PC上。

SP2的I/O子系统的结构如图8-24所示。SP2的I/O系统基本上是围绕着HPS建立的，并可以用一个LAN网关同SP2系统外的其他计算机连接。SP2的节点可以有4种配置。它们是宿主节点、I/O节点、网关节点和计算节点。宿主节点用来处理各种用户注册会话和进行交互处理；I/O节点主要用来实现I/O功能，如作为全局文件服务器。网关节点用于连网功能；计算节点专供计算。这4种节点也可能有重叠，例如一个宿主节点也可以

作为计算节点，一个 I/O 节点也可以作为网关节点。此外，SP2 还可以有一台到几台外部服务器，它们是在 SP2 之外附加的其他机器，如可以附加文件服务器等。

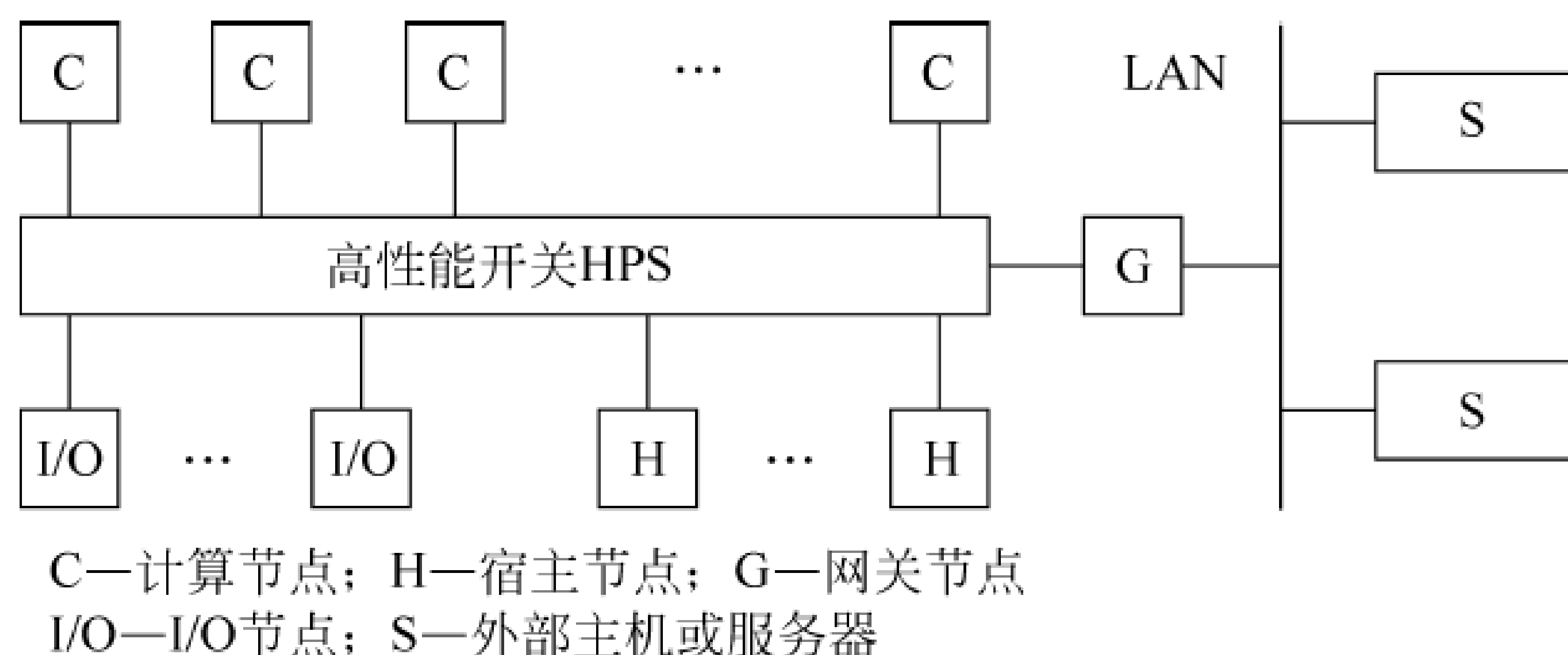


图 8-24　SP2 I/O 子系统结构

SP2 系统软件的核心是 AIX 操作系统。SP2 系统中，在 RS/6000 工作站原有环境下开发的软件大部分都能重用，包括各种串行的应用程序、数据库管理系统(DB2)、连机事务处理监控程序、系统管理和作业管理软件、FORTRAN、C++ 编译程序及标准的 AIX 操作系统等。SP2 系统只需添加一些作为一个可缩放的并行系统所需的新软件，或者对现成的软件做一些改进。

SP2 有一个集中的系统控制台用以管理整个系统。系统控制台用的是一台控制用的工作站，它可以使系统管理人员从单一地点对整个 SP2 系统进行管理。此外，在 SP2 硬件中，每个节点、每个开关和每个机架都有一个监视板，这种监视板能对环境条件进行检测，并对硬件部件进行控制。管理人员可以用这套设施来启动和切断电源，进行监控，把单个节点和开关部件置为初始状态。

8.5　多核处理器

多核处理器以其高性能、低功耗优势正逐步取代传统的单处理器成为市场的主流。本节介绍多核处理器产生的背景，Hydra、Cell、RAW 这三种典型的多核处理器结构，重点讨论核心结构选择、存储结构设计、片上通信、低功耗、软件应用开发 6 个影响当前多核处理器发展的关键技术，最后展望多核处理器的未来发展趋势。

8.5.1　多核处理器产生的背景

随着微电子技术的发展，现代计算机的核心部件——中央处理器(CPU)的晶体管数目和速度一直按照摩尔定律发展。过去三十多年里，CPU 设计者主要通过提高时钟速度、优化指令执行和增大片内高速缓存三个方面来提高 CPU 性能。

首先，提高时钟速度就是提升 CPU 工作的节拍，让 CPU 跑得更快，也就意味着让同样的工作能够在更短的时间里完成。其次是优化指令执行效率，尽量在同样的时间内完成更多工作，现代 CPU 设计中，一些指令的执行被不同程度地做了优化，如流水线设计、分支预测、同一时钟周期内执行更多指令，甚至指令流再排序支持乱序执行等。第三个是增大片内高速缓存(即 cache)，由于内存一直比 CPU 慢很多，因此让数据尽可能靠近 CPU，最好就是

直接放在 CPU 片内了，因此，片内缓存容量持续飙升了很多年。

然而，CPU 频率越高，所需电能和发热量也越多；制造工艺的进步，让晶体管更小，虽然耗电和发热量可能减少，但是造成了严重的电流泄漏问题，随之带来的就是大量的电能消耗和废热。允许并行执行的典型指令流存在一定的数量限制；在大多数应用情形下，超标量 CPU 同时并发 4 条以上指令时只能使性能增加一点点。另外，增加更多的管线寄存器以及一些旁路复用开关，并且防止流水线差错导致的性能损失，在电路上所能够增加的措施已经快到极限了。

由于实际散热限制出现的“功耗墙”，超标量发射的指令并行机制以及实际流水线长度的限制，使 CPU 在传统的架构上按摩尔定律来进一步提升性能对 CPU 设计者而言已经越来越难了。这种限制导致主要的 CPU 生产厂家如 Intel 等调整了它们的策略，不再是仅仅关注单 CPU Core 的时钟速率与处理性能。

CPU 设计者必须找到新的办法有效地提升晶体管数量，同时又降低功耗和设计复杂性。一个可行的办法是 CMP(Chip Multi-Processor)——多核 CPU 架构，即一颗芯片内集成多个处理器核。

8.5.2 多核处理器结构

多核处理器也称为片上多处理器(Chip Multi-Processor，CMP)，或单芯片多处理器。自 1996 年美国斯坦福大学首次提出片上多处理器(CMP)思想和首个多核结构原型，到 2001 年 IBM 推出第一个商用多核处理器 POWER4，再到 2005 年 Intel 和 AMD 多核处理器的大规模应用，最后到现在多核成为市场主流，多核处理器经历了十几年的发展。在这个过程中，多核处理器的应用范围已覆盖了多媒体计算、嵌入式设备、个人计算机、商用服务器和高性能计算机等众多领域，多核技术及其相关研究也迅速发展，如多核结构设计方法、片上互连技术、可重构技术、下一代众核技术等。

多核处理器将多个完全功能的核心集成在同一个芯片内，整个芯片作为一个统一的结构对外提供服务，输出性能。多核处理器首先通过集成多个单线程处理核心或者集成多个同时多线程处理核心，使得整个处理器可同时执行的线程数或任务数是单处理器的数倍，这极大地提升了处理器的并行性能。其次，多个核集成在片内，极大地缩短了核间的互连线，核间通信延迟变小，提高了通信效率，数据传输带宽也得到了提高。再者，多核结构有效共享资源，片上资源的利用率得到了提高，功耗也随着器件的减少得到了降低。最后，多核结构简单，易于优化设计，扩展性强。这些优势最终推动了多核的发展并逐渐取代单处理器成为主流。

在整体结构设计上多核处理器与传统的单处理器相比，多核内部结构没有固定的组织形式，可以有很多种实现方式。各个研究机构和厂商根据自己的应用目标设计出结构完全不同的多核结构。虽然如此，但在已有的多核处理器中仍存在几种比较典型的结构，它们分别代表了多核处理器结构中的某一类特点，而 Hydra、Cell 和 RAW 处理器就是三种典型的结构。

1. Hydra 处理器

Hydra 处理器是 1996 年美国斯坦福大学研制的集成了 4 个核心的处理器，这在当时是

一种新型的处理器结构。

Hydra 在一个芯片上集成了 4 个核心，核心间通过总线结构共享片上二级缓存、存储器端口和 I/O 访问端口，整体结构如图 8-25 所示。4 个核心采用了通用的百万指令级(MIPS)处理器，每个独立的处理核心有私有的一级缓存，其中指令缓存和数据缓存相互分离。4 个核心共享的二级缓存，采用 DRAM 存储。核心之间、核心到二级缓存、主存与片内以及 I/O 设备与片内的通信都是由总线结构来实现的。Hydra 被认为是一种典型的多核结构，不仅在于它是第一个多核处理器设计原型，还因为它采用了共享二级缓存的同构对称设计和基于高速总线的核间通信方式。

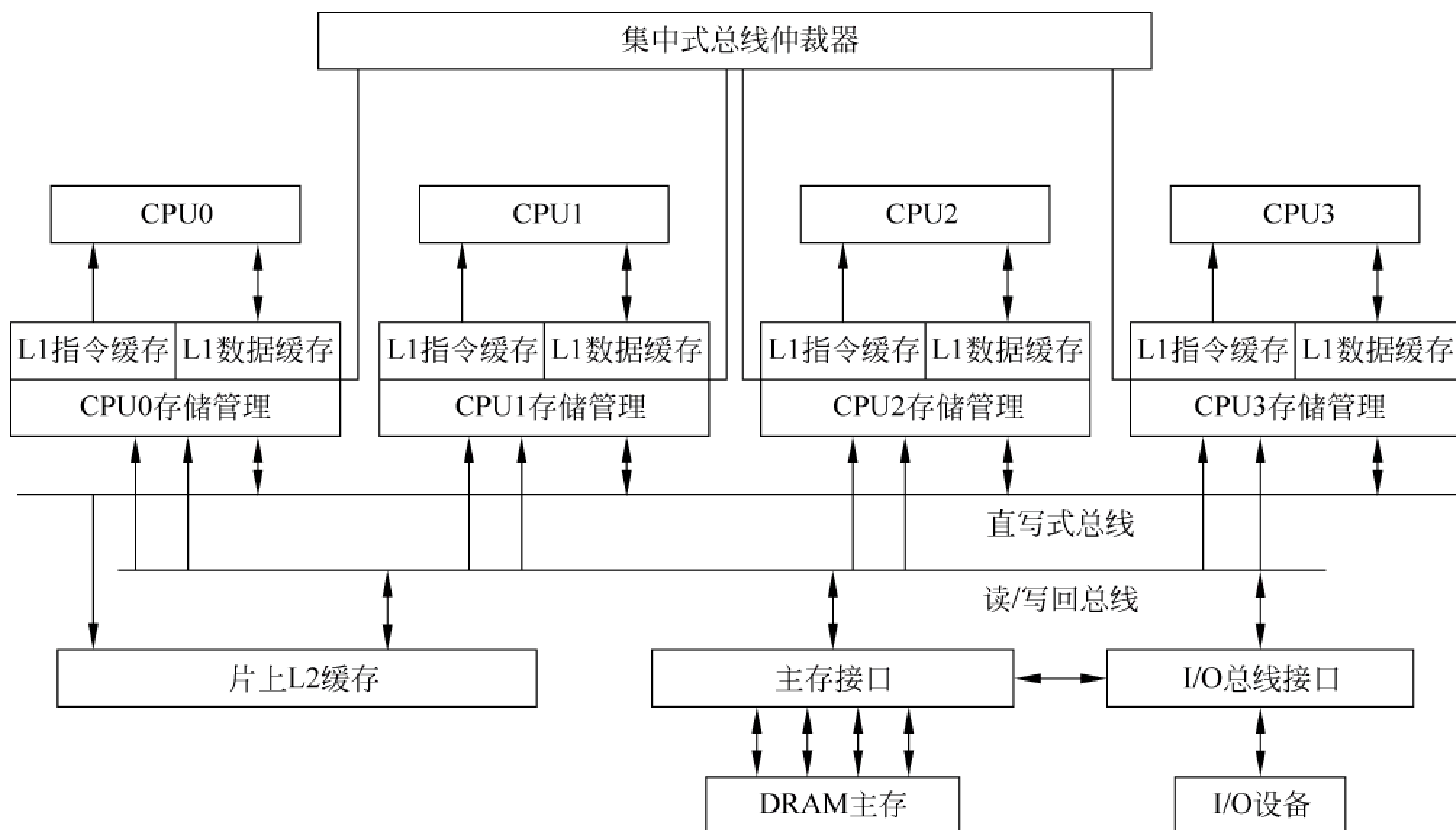


图 8-25 Hydra 处理器结构

2. Cell 处理器

2001 年 3 月，IBM 与 Sony、Toshiba 合作，着手开发一种全新的微处理器结构 Cell 处理器，旨在以高效率、低功耗来处理下一代宽带多媒体与图形应用。Cell 处理器的内部结构如图 8-26 所示。

Cell 处理器主要包含 9 个核心、一个存储器控制器和一个 I/O 控制器，片上的部件互联总线将它们连接在一起。核间通信和访问外部端口均是通过内部总线进行的，而且为了便于核间通信，整个 Cell 内部采用统一编址。这 9 个核心由一个 PowerPC 通用处理器(Power Processing Element，PPE)和 8 个协处理器(Synergistic Processing Element，SPE)组成。PPE 是一个有二级缓存结构的 64 位 PowerPC 处理核心。SPE 是一个使用本地存储器的 32 位微处理器，它没有采用缓存结构。PPE 与 SPE 除了在结构上不同外，它们的功能也有差别，PPE 是通用微处理器，拥有完整的功能，主要职能是负责运行基本程序和协调 SPE 间任务的运行；SPE 则是结构较简单，只用来从事浮点运算。Cell 的这种不对称结构被认为是一种典型的异构多核结构。

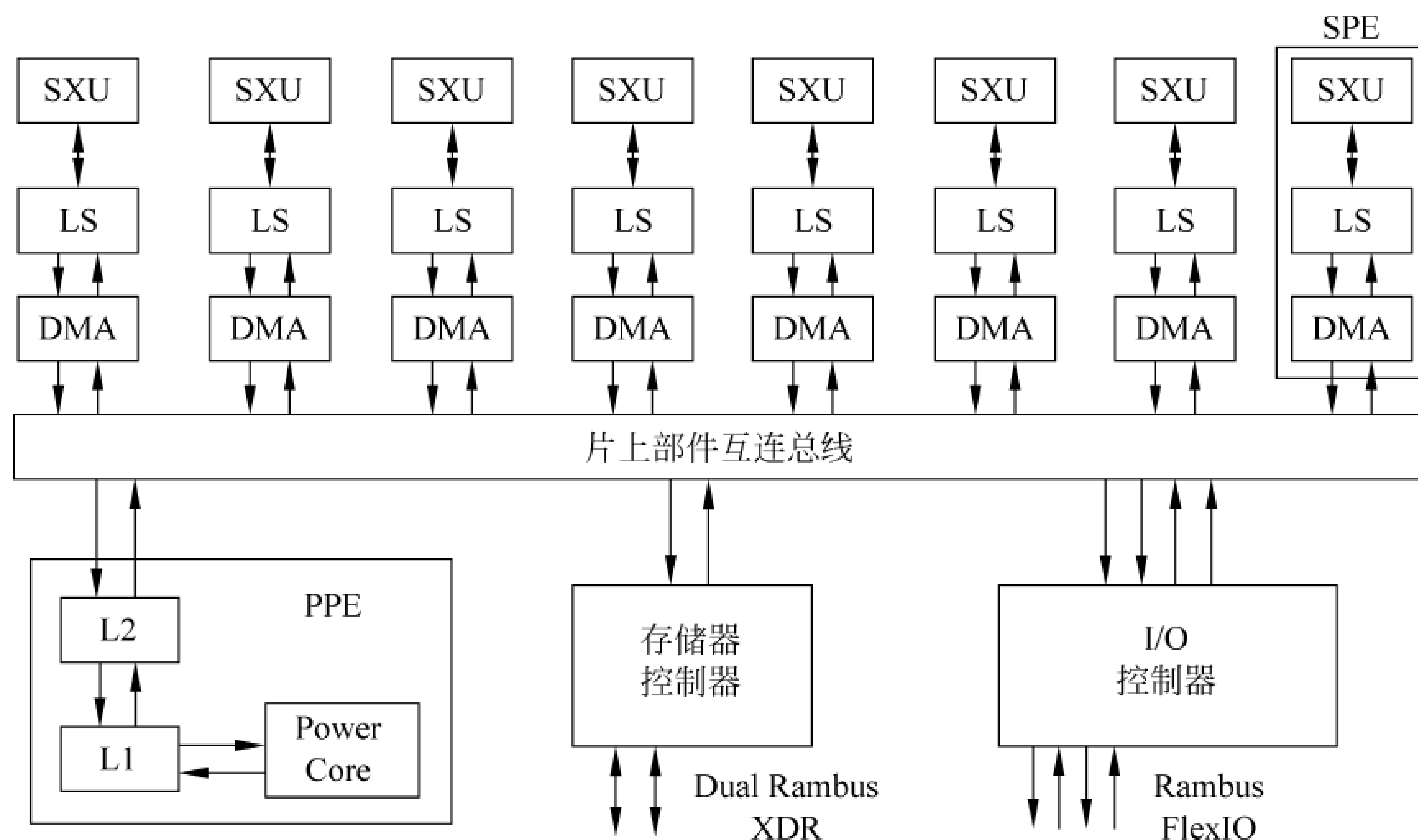


图 8-26 Cell 处理器结构

3. RAW 处理器

美国麻省理工学院研究的可重构 RAW 处理器芯片采用了一种 Tile 结构的多核处理器发展思路。RAW 处理器结构主要由 16 个 Tile 单元和片上网络构成，如图 8-27 所示。片上的每个 Tile 单元是一个完整的计算核心，负责处理各种运算，它包含了处理器、浮点运算单元、指令和数据缓存、静态和动态路由结构。片上网络是由可编程的、高度集成的互连结构构成的。

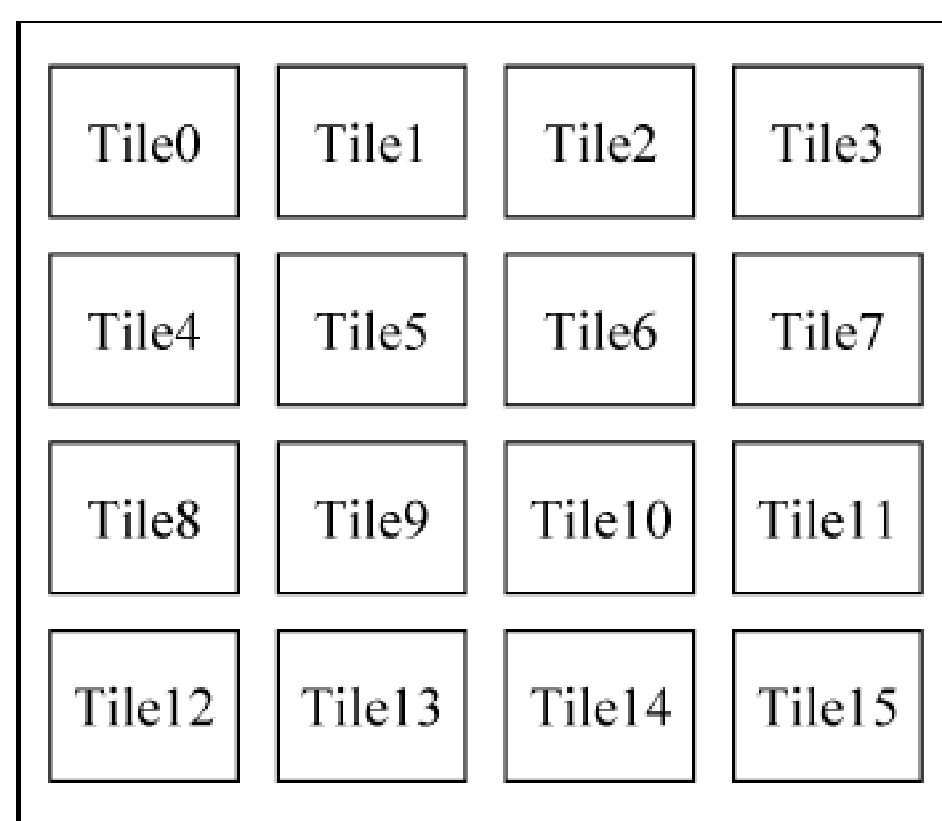


图 8-27 RAW 处理器结构

互连结构中的同步网络端口，让 Tile 间的访问通信延迟很小，性能接近寄存器的访问存取。RAW 中的 Tile 单元还能通过片上网络和芯片边缘的逻辑通道，高速简单地连接到外部存储器资源和各种 I/O 设备，可直接执行存储器存取。由于 RAW 处理器结构简单，功耗小，可扩展，而且片上网络通信效率高，因此 RAW 结构被认为是片上网络多核结构的代表。

8.5.3 多核发展的关键技术

多核处理器结构不仅有性能潜力大、集成度高、并行度高、结构简单和设计验证方便等诸多优势，而且它还能继承传统单处理器研究中的某些成果，例如同时多线程、宽发射指令、降压低功耗技术等。但多核处理器毕竟是一种新的结构，在多核结构设计和应用开发中出现了以前未曾遇到的新问题，这些问题给多核处理器的未来提出了挑战。

1. 核心结构的选择

多核处理器的核心结构主要有同构和异构两种。同构结构采用对称设计，原理简单，硬

件上较易实现。当前主流的双核和四核处理器基本上都采用同构结构。同构设计的问题在于：随着核心数量的不断增多，如何保持各个核心的数据一致；如何满足核心的存储访问和I/O访问需求；如何选择一个各方面性能均衡、面积较小以及功耗较低的处理器；如何平衡若干处理器的负载和任务协调等。

与同构结构相比，异构的优势是通过组织不同特点的核心来优化处理器内部结构，实现处理器性能的最佳化，而且能有效地降低功耗。但是异构结构也存在着一些难点。首先，搭配哪几种不同的核，核间任务如何分工以及如何实现。其次，结构是否具有良好的扩展性，还是受到核心数量的限制。再者，处理器指令系统设计和实现也是问题。因为不同核所用的指令系统对系统的实现也是很重要的，那么采用这些不同的核，是采用相同的指令系统还是不同的指令系统，能否运行操作系统等，也是需要考虑的内容。

2. 存储结构设计

处理器与主存储器之间的速度差距一直是处理器结构设计中必须考虑的问题，因为存储系统自身的体系结构设计直接关系到系统的整体性能，会对整个芯片的尺寸、功耗、布局、性能以及运行效率等各方面产生很大的影响。以往在单处理器中通过采用缓存结构基本上能较好地解决这个问题，能保证处理器性能得到发挥。可是，发展到了多核处理器时代，核心和主存之间因速度差距而带来的问题变得严重了。由于处理器内部核心数目增多，对主存的访问需求增加，而单处理器时代的缓存层次和访问带宽已经不能跟上多核处理器的访问需求，必须针对多核处理器进行相应的存储结构设计，并解决好存储系统的效率问题。

对存储系统设计，绝大多数处理器采用缓存设计，也有些处理器采用了片上存储器结构。缓存结构设计的优点是硬件设计与实现容易，易于应用开发与编程，缺点是需要保证缓存数据的一致，而且结构扩展不易。针对缓存数据一致性问题，其解决策略主要有总线侦听协议和基于目录的目录协议。侦听协议是每块缓存通过缓存侦听器时侦听总线，以接受一致性命令，不足的是它只适合核心数目较少的情况。目录协议是通过目录表记录自身存储块在其他缓存中的状态，以便维持一致性时使用点对点的通信，缺点是实现代价太大，并发访问目录时存在性能瓶颈。除了上述的硬件一致性算法，还有基于多处理机的软件一致性算法，但能否作为多核结构的缓存一致性机制，这些需要进一步的探讨研究。目前大多数多核处理器采用总线的侦听协议。

片内存储器是将片外的存储器引到片内，它与片外存储器一样是统一编址的，因此它避开了缓存不命中和一致性问题，但由于采用了存储器结构，其访问延迟较缓存大。当前一些研究人员通过采用高速动态随机存储器来组成片内存储器，缩小了与缓存间的性能差距。

除了选择何种存储结构外，存储结构设计的问题还有：存储器多大比较合适；在哪一级实现数据的共享和通信比较合适；在哪一级解决缓存一致性问题比较合理；存储结构如何支持多线程的应用等。

3. 片上通信

多核芯片上的多个核心虽然各自执行自己的代码，但是不同核间可能需要进行数据的共享和同步，因此片上通信结构的性能将直接影响处理器的性能。当前，片上通信主要有三种方式：总线共享、交叉开关互连和片上网络(Network On Chip，NOC)。

总线共享结构是指片上核心、输入输出端口以及存储器通过共享二级或三级 cache，或者通过连接核心的总线进行通信。总线结构的长处是较为简单，易于设计实现，当前多数双核和四核处理器基本上都采用了该结构，但缺点是总线结构可扩展性较差，适用于核心数较少的情况。比较典型的总线共享结构处理器有 Hydra、Intel 的 Core、IBM 的 Power4/5 等。

交叉开关互连结构由交叉开关以及接口逻辑构成。交叉开关与总线结构相比，优势是数据通道多，访问带宽更大，但不足是交叉开关结构占用的片上面积也较大，而且随着核心数的增加，性能也会下降，因此它也只适用于核心数较少的情况。例如 AMD 公司的 Athlonx2 双核处理器用交叉开关来控制核心与外部的通信。

片上网络把互联网络用于片上系统设计，解决片上组件之间的通信问题，它借鉴了并行计算机的互联网络。片上网络与并行计算机的互连有很多相同点：支持包通信、可扩展、提供透明的通信服务等；但也有不同之处：片上网络技术支持同时访问，而且有可靠性高以及可重用性高等特点。它与总线结构、交叉开关结构相比，可以连接更多 IP 组件、可靠性高、可扩展性强以及功耗较低，因此片上网络被认为是更加理想的大规模 CMP 互联技术。当前片上网络主要有二维网格网络、3KTours 等互联结构。片上网络设计的问题是寻找网络开销和多核耦合程度最佳的平衡，并同时考虑网络的可扩展性。RAW 处理器就采用了片上二维网络结构，它通过集成高速网络和优化的路由算法，片上核心间的通信延迟最大不会超过 6 个周期，而且该结构可扩展性强。

这三种结构虽各有优势和不足，但亦可融合，如在全局范围采用片上网络而在局部选择总线或者交叉开关结构，以实现性能与复杂性的平衡。

4. 低功耗

传统单处理器的一个瓶颈就是随着频率的提升，功耗越来越高，最终使得芯片无法正常运行。在早期的多核处理器设计中，主要通过降低核心频率来降低处理器的功耗，但是这样限制了核心的运算性能，并没有从根本上实现高性能、低功耗的目的。功耗过高不仅导致能源消耗，而且热堆积和过高的功耗密度也会对系统稳定性造成影响。现在一个芯片上可以集成 10 亿个晶体管，如此众多的片上资源，如何控制它的功耗，保持较高性能，成为了一个重要的问题。

在多核处理器产生以前，低功耗技术主要有降低动态消耗和降低静态消耗技术两方面。动态消耗包括处理器内部各元件正常工作时所消耗的电能，例如电容性的充放电、切换频率、逻辑门的状态转换等。降低动态消耗一直以来都是人们研究的重点，而且技术比较成熟。降低动态消耗技术现在主要有多元功能电压技术、动态电压调节、时钟屏蔽技术等。静态消耗技术是指来自漏电流的功率消耗，特点是即使元件处在空闲状态也会消耗电能，具体包括亚阈值漏电流和门漏电流。降低静态消耗技术的主要技术有通道长度调整、寄存器锁存技术、能量选通技术等。

上面两方面的技术主要在电路层次上去进行低功耗设计和技术开发。在多核处理器出现以前，这些技术便已出现在单核处理器上。随着多核处理器的产生，由于多核处理器在结构和实现上有了新的特点，所以研究人员又在新的方面发现了降低功耗的方法，例如异构结构设计、动态线程分派与转移技术等。异构的结构设计就是利用异构结构对片上资源的最佳化配置，处理器的执行效率提升，使得处理器不仅具有高性能也降低了功耗。动态线程分

派与转移技术是指利用多核心处理能力，将某个核心上过多的负载转移到负载小的核心上，从而使处理器在不降低处理性能的情况下，降低处理器功耗。

研究人员还通过对操作系统的设计和优化来降低多核处理器的运行功耗。例如当任务较少时，操作系统会关闭一个核心或降低处理器频率，并降低封锁转速，来使整个系统降低消耗。因此，低功耗设计包含了电路级、结构级、算法级和操作系统级等多个方面的内容，是一个需要从多方面进行综合考虑的问题。

5. 平衡设计原则

平衡设计原则是指在芯片的复杂度、内部结构、性能、功耗、扩展性、部件成本等各个方面做一定的权衡，即不能为了单纯地获得某一方面的性能而导致其他方面的问题，在设计过程中要坚持从整体结构的角度去权衡各个具体的结构问题。

在多核处理器设计工程中，需要坚持平衡设计的原则。因为往往在减少一个方面问题的同时就增加了另一个方面的问题，所以在设计过程中要仔细权衡对某些问题的解决方法，尽量采用简单、易于实现、成本低廉而且对整体性能影响不大的设计。微处理结构设计的重点不在于其中某一个细节采用什么复杂或性能表现较好的设计，而是在于整体的设计目标。也即是说要得到在一个通常情况下，逻辑结构简单和对大多数应用程序有良好性能的微处理器结构，在适当的时候为了整体目标就不得不牺牲某一方面。当然在具体的设计中，不能只是简单地选择，应该是建立在科学的实验和模拟分析基础上的选择或平衡。因此在多核处理器设计中，要以科学分析的数据结果为基础，坚持合理平衡的设计原则。

6. 软件应用开发

多核处理器在利用多个核心的并行执行能力来提高处理器运算性能的同时，也给软件开发者带来了麻烦。当前的困境是众多应用并没有利用多核的性能潜力，多核的性能优势没有体现。这主要有两个原因：一方面是并行编程复杂，没有一个易于程序员编写并行程序的编程工具和环境；另一方面是一些应用的线程级加速潜力有限。

并行编程困难的问题从并行计算机产生以来就存在，只是随着多核的主流化，问题更加突出了。多核系统下的并行编程，主要是开发多核的线程级并行性，但是已有的并行编程模式、编程语言不能完全适合多核环境，不能将多核的多线程并行潜力完全发挥出来，例如OpenMP、MPI、并行C等。因此，针对多核环境下对并行编程应用的要求，许多研究机构和公司一方面对现有的并行编程模型和编程语言进行修改和改进，例如改进支持共享存储结构的OpenMP、OpenMP＋MPI的混合编程模型、PThread多线程编程模型；另外一方面，在资金项目的支持和市场的推动下，各类研究机构也正在积极研制开发新一代的并行编程模型和并行编程语言，例如事务存储编程模型、Intel公司的C++编程模型、IBM公司的X10语言、Sun公司的Fortress语言、Cray公司的Chapel语言等。

另外一方面，多数应用的并行加速潜力有限是因为在实际应用中，可供程序员迅速开发出来的程序还是单线程的，同时多数应用也是用C或C++编写的，它们一直被设计为单线程，而单线程程序中所能开发和利用的并行性始终是有限的。所以对于这些应用，要么重新编写并行代码，要么研发面向多核结构的自动并行化工具，使得这些应用能在多核处理器系统中高效应用。

8.5.4 多核发展趋势

多核处理器产生的直接原因是替代单处理器，解决微处理器的发展瓶颈，但发展多核的深层次原因还是为了满足人类社会对计算性能无止境的需求，而且这种压力还会持续下去。即便在当前，设计者已经有效地将多核性能提高到了一个新的水平，可是人们对提升性能的渴望并未停顿。阻碍多核性能向更高水平发展的问题很多，可真正束缚多核发展的是低功耗和应用开发两个问题。由于现有的多核结构设计方法和技术还不能有效地处理好这两个问题，因此有必要在原有技术的基础上探索新的思路和方法。下面的内容是为了实现高性能、低功耗和高应用性的目标多核处理器呈现的几种发展趋势：

（1）多核上将集成更多结构简单、低功耗的核心。为了满足性能需求，通过集成更多核心来提高性能是必然选择，但是核心的结构也必须考虑。因为如果核心结构过于复杂，随着核心数量的增多，不仅不能提升性能，还会带来线延迟增加和功耗变大等问题。例如，2007年 Tilera 公司和 Plurality 公司分别推出自己的 64 核处理器产品，而 Intel 公司也将推出 80 个核心的低功耗处理器。

（2）异构多核是一个重要的方向。研究表明，将结构、功能、功耗、运算性能各不相同的多个核心集成在芯片上，并通过任务分工和划分将不同的任务分配给不同的核心，让每个核心处理自己擅长的任务，这种异构组织方式比同构的多核处理器执行任务更有效率，实现了资源的最佳化配置，而且降低了整体功耗。

（3）多核上应用可重构技术。大规模高性能可编程器件的出现，推动了现场可编程门阵列(Field Programmable Gate Arrays，FPGA)技术的发展。在芯片上应用 FPGA 技术有高灵活性、高可靠性、高性能、低能耗和低成本多种优势。微处理器设计人员注意到了这种优势，并将 FPGA 等可重构技术应用到多核结构上，让结构具备可重构性和可编程性。这种创新思路大大提高了多核的通用性和运算性能，使处理器既有通用微处理器的通用性，又有专用集成电路的高性能，使之兼具灵活性、高性能、高可靠、低能耗等众多优良特点。

知识拓展

超级计算机“天河二号”

“天河二号”是集群式超级计算机系统，由中国国防科学技术大学研制成功，其峰值计算速度为每秒 5.49 亿亿次、持续计算速度为每秒 3.39 亿亿次双精度浮点运算，性能优异。在 2014 年 6 月 23 日公布的全球超级计算机 TOP 500 强榜单中，中国“天河二号”的速度位居榜首，成为全球最快的超级计算机。

1）系统组成

“天河二号”由 16 000 个浪潮的节点组成，共有 170 个机柜，包括 125 个计算机柜、8 个服务机柜、13 个通信机柜和 24 个存储机柜，占地面积 720 平方米，最大运行功耗 17.8MW，具体技术参数如下。

处理器：16 000 个运算节点，每节点配备两个 Xeon E5 中央处理器、三个 Xeon Phi 57 核心的协处理器(运算加速卡)。累计 32 000 个 Xeon E5 主处理器和 48 000 个 Xeon Phi 协处理器，共 312 万个计算核心。

中央处理器为Intel公司提供的，运作时钟频率为2.2GHz的Xeon E5-2692 12核心处理器，基于Intel Ivy Bridge微架构(Ivy Bridge-EX核心)，采用22nm制程，峰值性能0.2112TFLOPS。

运算加速上，使用基于Intel集成众核架构的Xeon Phi 31S1P协处理器，运行时钟为1.1GHz，每个使用61个核心中的57个(因为使用61个会存在运算周期协调问题)，每核心借由特殊的超线程技术能运作4个线程，产生峰值性能为1.003TFLOPS。

内存：每个节点拥有64GB主存，而每个Xeon Phi协处理器板载8GB内存，故每节点共88GB内存，整体总计内存1.408PB。

外存：12.4PB容量的硬盘阵列。

主板、机架与机柜均由浪潮集团制造，共有125个计算机柜，每个机柜容纳4个机架，每个机架容纳16块主板，每个主板设置有两个计算节点。每块主板上分为APU模块和CPM模块两部分，APU部分承载5块Xeon Phi，CPM部分承载1块Xeon Phi+4颗Xeon E5。

APU模块和CPM模块之间以CPU内部提供的PCI-E 3.0 16x接口进行连接，但实际由于Xeon Phi的硬件限制，仅支持至PCI-E 2.0 16x，单通道数据传输速率为10Gb/s。

前端处理器：计算节点前端处理器为4096颗FT-1500 16核心SPARC V9架构的处理器，40nm制程，运作时钟频率1.8GHz，热设计功耗65W，峰值性能144GFLOPS。

连接：使用光电混合传输技术(Optoelectronics Hybrid Transport Technology)，使用自制的TH Express-2主干拓扑结构网络连接，以13个大型路由器通过576个连接端口以光电传输介质与各个运算节点互联，控制器名为NRC，使用90nm制程，单个控制器的数据吞吐量2.56Tb/s，终端网络接口使用名为NIC的控制器，以PCI-E 2.0接口连接，数据传送速率为6.36GB/s。

操作系统：麒麟操作系统、基于资源管理用单一Linux公用程序(Simple Linux Utility for Resource Management，SLURM)的全局资源管理。

能耗：整机功耗17 808kW，在搭载水冷散热系统以后，功耗将达到24MW，每瓦性能为1.901GFLOPS。

2) 关键技术

“天河二号”取得了一系列技术创新和突破，涉及体系结构、芯片技术、高速互连、存储架构、编程模型、能耗控制、系统管理、结构工艺等诸多方面，使得“天河二号”综合技术取得了国际领先地位。

“天河二号”基于新型异构多态体系结构，在强化科学工程计算的同时，可高效支持大数据处理、高吞吐量和高安全信息服务等多类应用需求；采用了微异构计算阵列和新型并行编程模型及框架，提升了应用软件的兼容性、适用性和易用性。“天河二号”服务阵列采用了国内主频最高的自主高性能通用CPU FT-1500，具有高吞吐量和高安全信息服务功能。基于自主通信接口芯片和互联交换芯片设计实现了光电混合的自主定制高速互联系统，性能是当前国际上最先进的商用互联系统的2倍。采用综合化的能耗控制机制，能效比进入国际先进行列。设计实现了基于背板前后对插、水平盲插的高密度高精度组装结构，使得“天河二号”的计算密度处于国际领先水平。

3) 应用领域

“天河二号”已应用于生物医药、新材料、工程设计与仿真分析、天气预报、智慧城市、电

子商务、云计算与大数据、数字媒体和动漫设计等多个领域,还将在生命科学、材料科学、大气科学、地球物理、宇宙、经济学,以及大型基因组组装、基因测序、污染治理等一系列事关国计民生的大科学、大工程中"大显身手"。此外,国家超算广州中心积极推动国际交流与合作,利用"天河二号"为国外研究机构提供高性能计算服务。

超级计算机的应用涉及各行各业,需要很多学科的深度学科背景和专业人才,才能将该领域的大型软件开发出来。因此与美国等一些发达国家相比,我国在产业能力和应用水平等方面还有差距。

本章小结

本章主要讲述了以下内容:

(1) 计算机系统的并行性。介绍了计算机体系结构的概念,体系结构中的并行性;介绍了时间重叠、资源重复和资源共享三种提高并行性的主要技术途径;并对并行计算机体系结构的 Flynn 分类法进行了讲述。

(2) 流水线技术。介绍了流水线的定义、分类、主要性能参数(吞吐量、加速比和效率)和流水线中产生的相关问题;结合例子分析了如何进行流水线调度及流水线调度方案的选取方法;并介绍了超流水线处理机和超标量超流水处理机。

(3) 多处理机系统。介绍了多处理机的主流 SMP 和 DSM;讲述了多处理机的 cache 一致性问题和多处理机操作系统及多处理机的并行性的实现。

(4) 机群系统。介绍了机群系统的定义、组成及其关键技术。

(5) 多核处理器。介绍了多核处理器的定义、典型结构、影响多核处理器发展的关键技术,展望了多核处理器的发展趋势。

习题八

一、名词解释

1. 同时性　　2. 并发性　　3. 时间重叠
4. 资源重复　　5. 相关　　6. SISD

二、选择题

1. 从执行程序的角度看,以下并行性等级最高的为(　　)。
   A. 指令内部并行　　B. 指令级并行
   C. 线程级并行　　D. 任务级或过程级并行
2. 从处理数据的角度,以下并行性等级最高的为(　　)。
   A. 字串位串　　B. 字串位并　　C. 字并位串　　D. 全并行
3. 按照 Flynn 分类法,以下(　　)计算机系统属于真正的单处理机系统。
   A. SISD　　B. SIMD　　C. MISD　　D. MIMD
4. 流水线基于的是下列(　　)并行技术。
   A. 时间重叠　　B. 资源重复　　C. 资源共享　　D. 都不是

5. 以下不属于流水线相关的是(　　)。
   A. 结构相关　　B. 执行相关　　C. 数据相关　　D. 控制相关

三、综合题

1. 什么是计算机体系结构?

2. 计算机系统结构和计算机组成及计算机实现三者的定义以及相互的关系如何?

3. 并行性的概念及其包含的含义是什么?并行性开发的途径有哪些?

4. 简述紧耦合和松耦合的多机系统结构。

5. Flynn 分类法中的数据流指的是什么?多倍性指的是什么,具体的分类如何?

6. 上网查询有关 Pentium 的资料,简述 Pentium 微处理器的基本结构;Pentium 微处理器的整数流水线是怎样工作的?

7. 动态流水线和静态流水线的区别。

8. 衡量流水线处理机性能的指标主要是什么?它们又是如何定义的?流水的最大吞吐量指的是什么?

9. 流水线的吞吐量受限于瓶颈过程,那么消除瓶颈过程对流水线吞吐量的影响的方法有哪些?

10. 延迟转移的思想是什么?

11. 分支转移预测功能的意义是什么?试举例说明。

12. 有一个 5 个功能段组成的乘-加双功能的静态流水线,乘由 1→2→3→4 完成,加由 1→4→5 完成,各段延时均为 Δt,输出可直接返回输入或存入缓冲器缓冲。现要求计算长度均为 6 的 A、B 两个向量逐对元素求和的连乘积。

$$S=\prod_{i=1}^{6}(A_i+B_i)$$

(1) 画出流水线完成此运算的时空图;

(2) 求出顺序执行和采用流水方式执行两种情况下分别各需要多少个 Δt 时间;

(3) 计算该流水线的吞吐量、加速比和效率。

13. 有一个 4 段 6 拍的单功能非线性流水线,其预约表如表 8-1 所示。

表 8-1　预约表 1

段号＼时钟	t_1	t_2	t_3	t_4	t_5	t_6
S_1	√					√
S_2		√	√			
S_3				√		
S_4					√	

(1) 求出其冲突向量、延迟禁止表,并画出其状态转移图;

(2) 求出其流水线的最佳调度方案及此时的最大吞吐量。

14. 有一个 4 段 6 拍的单功能非线性流水线,其预约表如表 8-2 所示。

表 8-4 预约表 2

段号＼节拍	1	2	3	4	5	6
S_1	*					*
S_2		*	*			
S_3				*		
S_4					*	

（1）求出其冲突向量 **C**、延迟禁止表 F，并画出其状态转移图；

（2）求出其流水线的最佳调度方案及此时的最大吞吐量；

（3）如果按照(2)求出的最佳调度方案送入 6 个任务进入流水线，画出此时的流水线时空图，并求出其吞吐量、加速比和实际效率。

15. 某带双输入端的加、乘双功能静态流水线有 1、2、3、4 四个子部件，延时分别为 Δt、Δt、$2\Delta t$、Δt，其中"加"功能由 1→2→4 组成，"乘"由 1→3→4 组成，输出可直接返回输入或锁存。现欲执行

$$\sum_{i=1}^{4}[(ai+bi)*ci]$$

（1）画出此流水时空图，并标出流水线入端数据变化的情况；

（2）计算此运算全部完成所需的时间及在此期间流水的效率、吞吐量、加速比；

（3）将瓶颈部分再细分，画出解此题的时空图；

（4）求出按(3)解此题所需的时间及在此期间流水线的效率、吞吐量、加速比。

16. 简述什么是超流水线处理机和超标量超流水处理机。

17. 简述什么是多处理机系统。

18. 简述解决多处理机中 cache 之间的一致性问题的基本方案有哪些。

19. 举例说明多处理机操作系统有哪几种？并进行简单的比较。

20. 简述监听协议——MESI 协议的 4 种状态的含义。

21. 多处理机系统相对于单处理机系统而言，两者在并行性的实现上有何区别与联系？

22. 多处理机中的并行性表现在哪些方面？开发多处理机的并行性有哪些途径？

23. 由霍纳法则给定的表达式为 $E=a(b+c(d+e(f+i(j+k))))$，利用降低树高的办法来加速运算，那么：

（1）画出树状流程图；

（2）确定 T_P、P、S_P、E_P 的值。

24. 简述并行算法中为提高并行度而对树状结构进行变换的过程。

25. 什么是机群系统？机群中采用单一系统映像(SSI)技术的目的是什么？

26. 多处理机系统与并行处理机的主要差别是什么？多处理机系统主要解决的技术问题是什么？

参 考 文 献

[1] John L Hennessy. Computer Architecture A Quantitiative Approach(Fifth Edition). New Jersey: Morgan Kaufmann,2011.

[2] William Stallings. Computer Organization & Architecture (Ninth Edition). New Jersey: Person Education,Inc.,2013.

[3] Linda Null. The Essentials of Computer Organization and Architecture. Sudbyry: Jones and Bartlett Publishers, Inc., 2006.

[4] John D Carpinelli. Computer System Organization and Architecture. New Jersey: Person Education, Inc.,2003.

[5] 王爱英. 计算机组成与结构. 5版. 北京:清华大学出版社,2013.

[6] 白中英. 计算机组成与系统结构. 5版. 北京:科学出版社,2011.

[7] 张晨曦. 计算机系统结构. 北京:高等教育出版社,2008.

[8] 李学干. 计算机系统结构. 5版. 陕西:西安电子科技大学出版社,2011.

[9] 汤小丹. 计算机操作系统. 4版. 陕西:西安电子科技大学出版社,2014.

[10] 中国计算机科学与技术学科教程研究组. 中国计算机科学与技术学科教程. 北京:清华大学出版社,2002.